服务计算
Service Computing

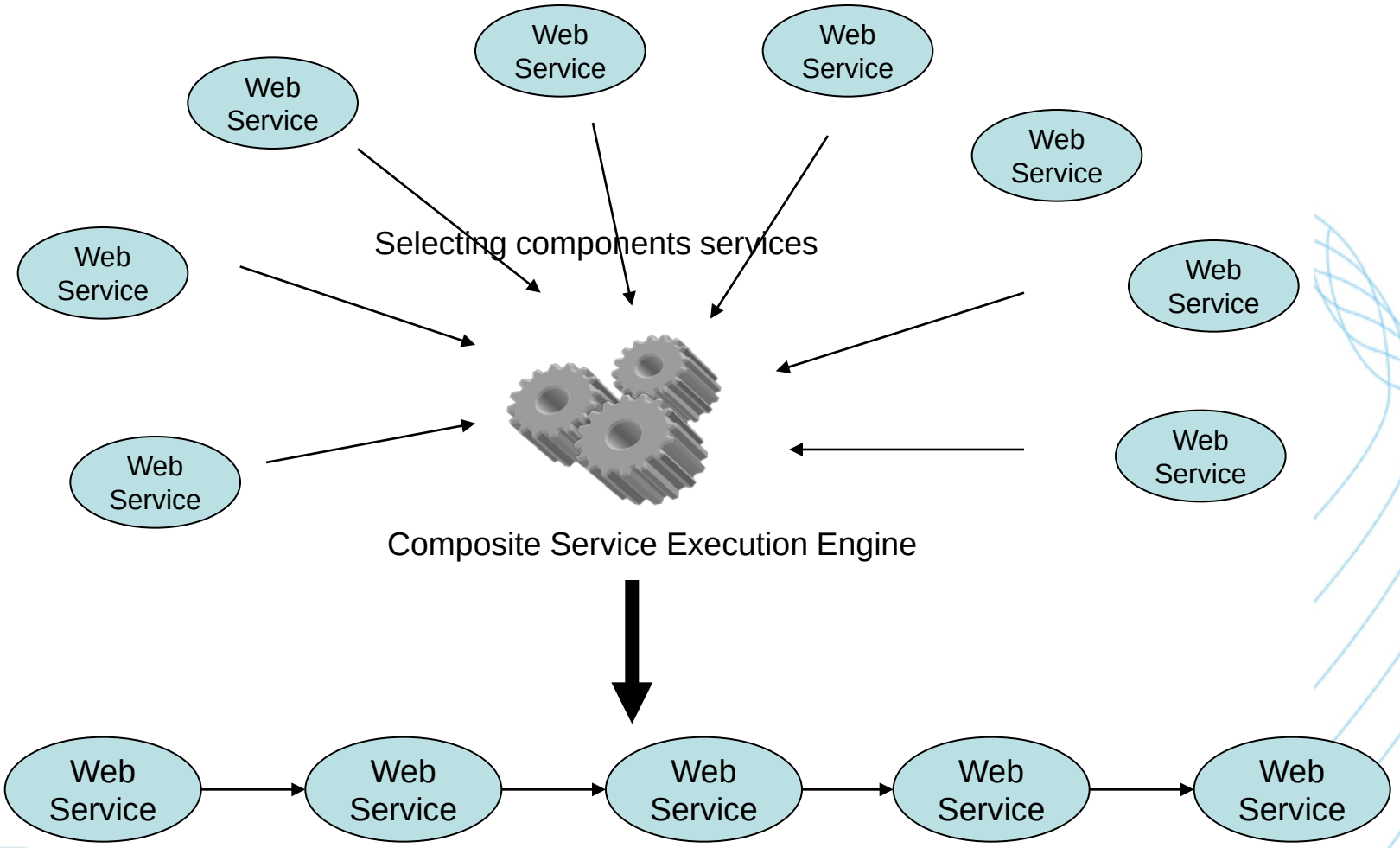
王旭
wangxu@buaa.edu.cn

六、服务组合

SOA设计的初衷：重用与集成

- 重用小粒度的服务，将小粒度服务集成为大粒度服务 (compose fine-grained services into coarse-grained service)
- 将硬编码的集成变为动态可配置的集成 (transform hard-coded integration into dynamically reconfigurable integration)
- SOA的优势所在，也是追求的目标所在

服务组合



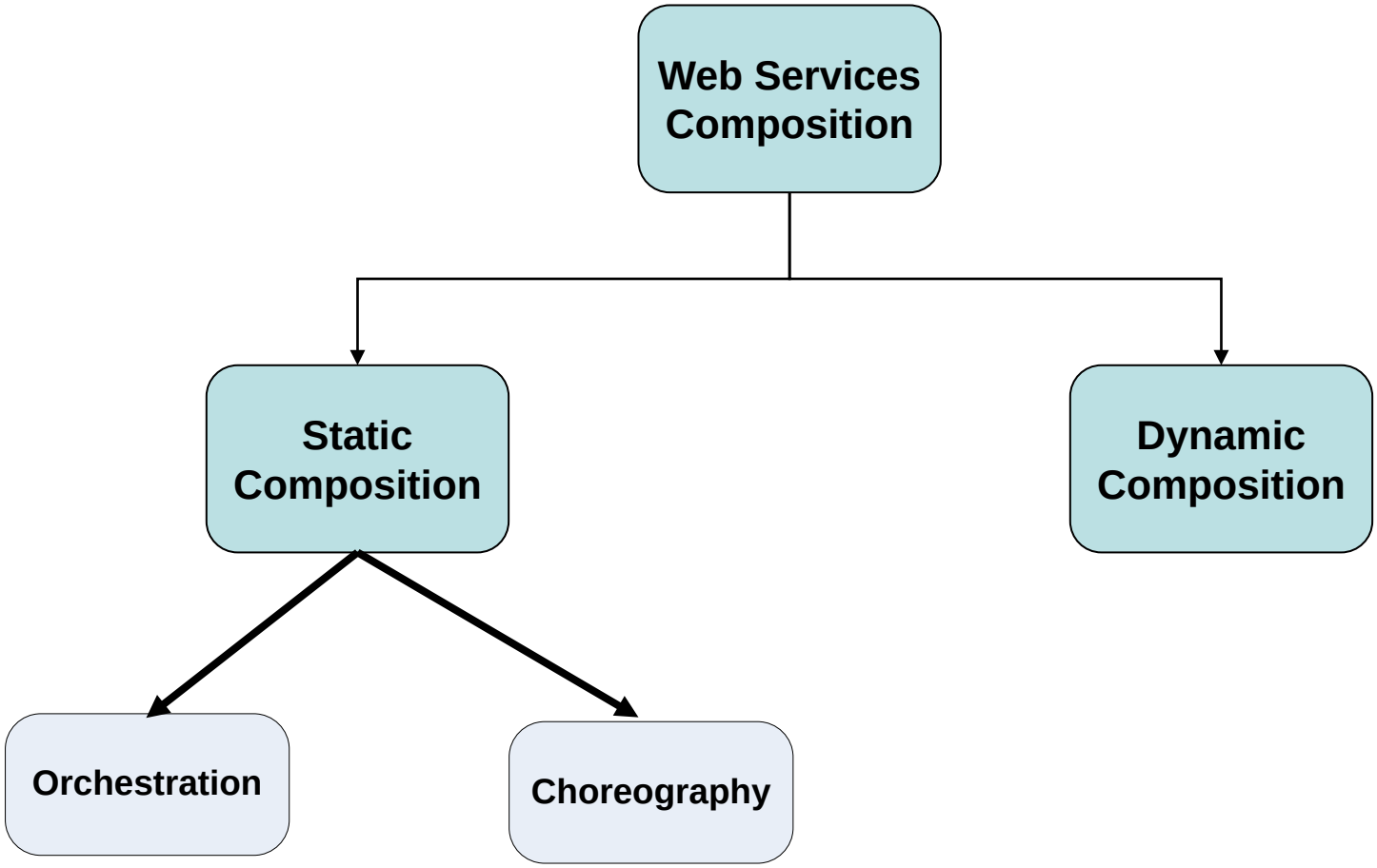
服务组合的类型

- Web服务组合
  - 手动：静态组合
  - 半自动：动态组合
  - 全自动组合
- RESTful服务/Web API组合

服务组合的类型

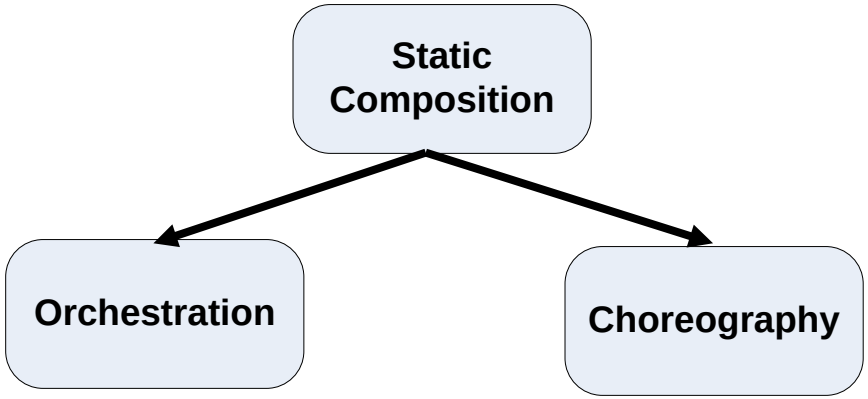
- Web服务组合
  - 手动：静态组合
  - 半自动：动态组合
  - 全自动组合
- RESTful服务/Web API组合

Web服务组合



静态组合

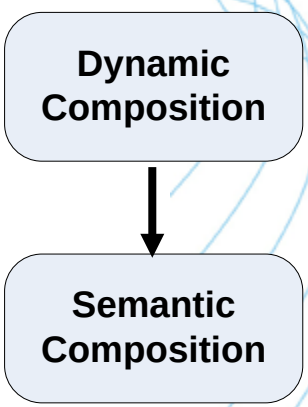
- 也称基于句法(Syntactic)的组合
  - 规定：服务被调用的顺序；某个服务不被调用的条件



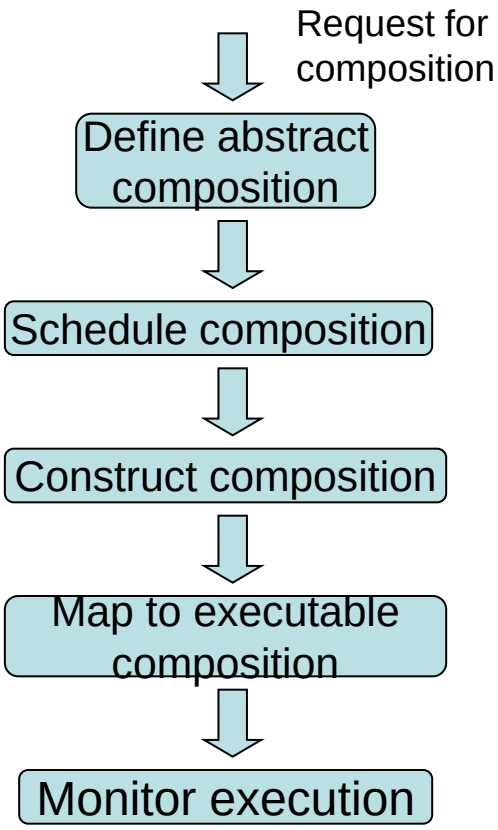
  - **Orchestration(like BPEL4WS)**
    - 通过一个中心的协调器将可用的Web服务组合起来
    - Orchestrator负责调用和组装Web服务
  - **Choreography(like WS-CDL)**
    - 无中心协调器
    - 复杂任务通过参与服务间的会话(Conversation)来定义
    - 是一种协同服务间对等交互(Interaction)的组合

动态组合

- **也称基于语义的服务组合**
  - 静态组合：信息流程和服务之间的绑定在设计时已知
  - 动态组合：运行时动态发现、组合、绑定、调用
  - 基于语义的技术
    - RDF+Ontologies
    - OWL-S and WSMO



动态组合



定义: 规定活动、约束、需求、可能的异常行为
调度：服务何时如何执行
构建：根据前阶段结构，系统构造Web服务组合的一个/多个可执行模式
执行：运行可执行模式
监控：监控执行过程

Web服务组合的特征

- 连接性Connectivity
  - 可靠性Reliability
    - 提供时间上连续服务响应的能力
    - 在两个端点间正确分发消息的能力
  - 可访问性Accessibility
    - 对每个服务请求响应的概率
  - 异常处理/补偿
    - 错误产生的后果，如何回滚已完成的活动
    - 失效时，Web服务调用的补偿管理能力

Web服务组合的特征

- 正确性Correctness:
  - 安全性/活性 Safety/Liveness
    - 计算过程中，不会发生bad事件
    - 计算过程中，某些事件最终会发生
  - 安全性/可信 Security/Trust
    - Web服务提供适当认证，授权，保密性和数据加密的能力
    - Web服务在环境破坏、人为错误、恶意攻击、设计/实现错误情况下，如期执行的能力

Web服务组合的特征

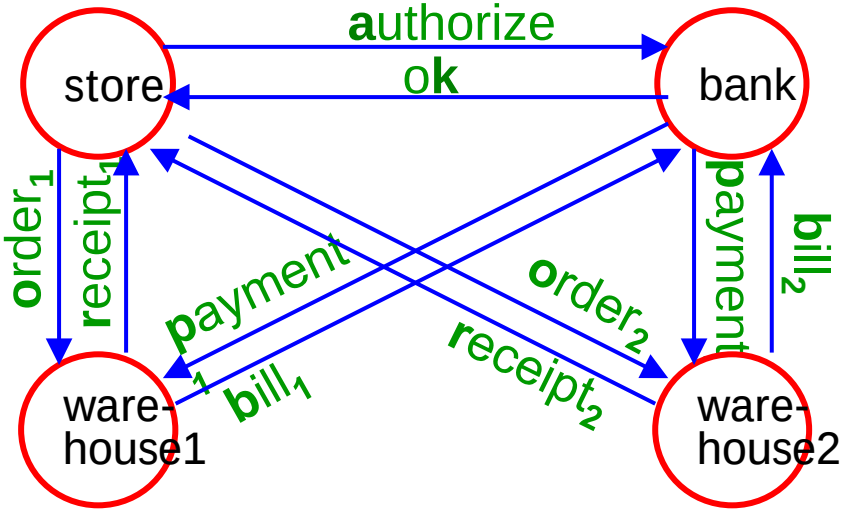
- 服务质量 (QoS)
  - 准确性 Accuracy
    - Web服务的错误率：某时间段内服务产生的错误数
  - 可用性 Availability
    - 一个服务在给定时间内的可用概率：在延长时间内可用的时间百分比
  - 性能 Performance
    - 服务请求的成功率
      - 处理请求的最大时间（响应时间）
      - 一段时间内处理的请求数（吞吐率）
      - 处理请求的需要时间（延迟）

静态服务组合的两大概念

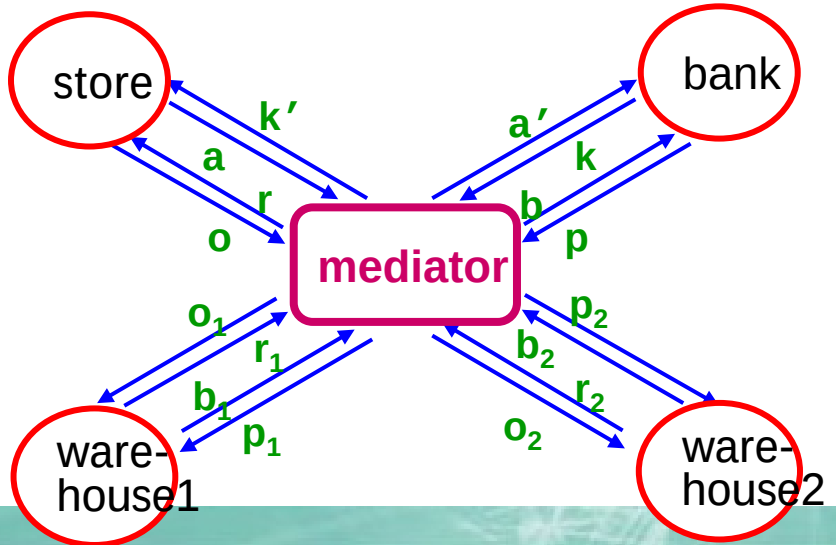
- SOA中提出了两个概念：orchestration和choreography
  - 前者定义了如何将小粒度的服务按照特定的流程聚合为大粒度的服务；
  - 后者则定义了如何在多方的业务流程之间通过服务实现协同的动作编排。
- 二者的本质上都是用来规划服务之间的协同。
- 目的
  - 编制：描述和执行一个局部(组织)的内部流程模型
  - 编排：描述和指导一个全局模型—组织间协作模型
- Choreography模型向Orchestration模型转换

Orchestration vs. Choreography

- **Peer-to-peer**

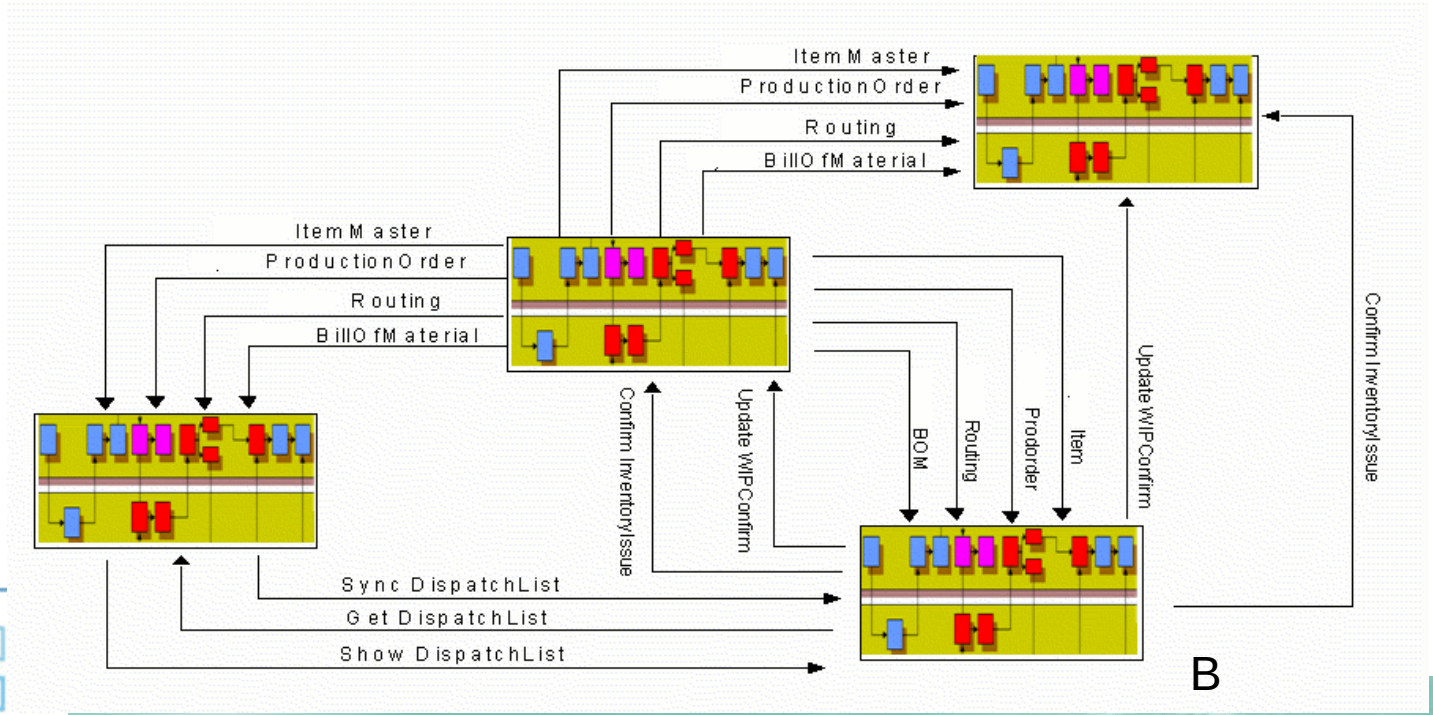


- **Mediated, or “hub and spoke”**



Choreography与orchestration

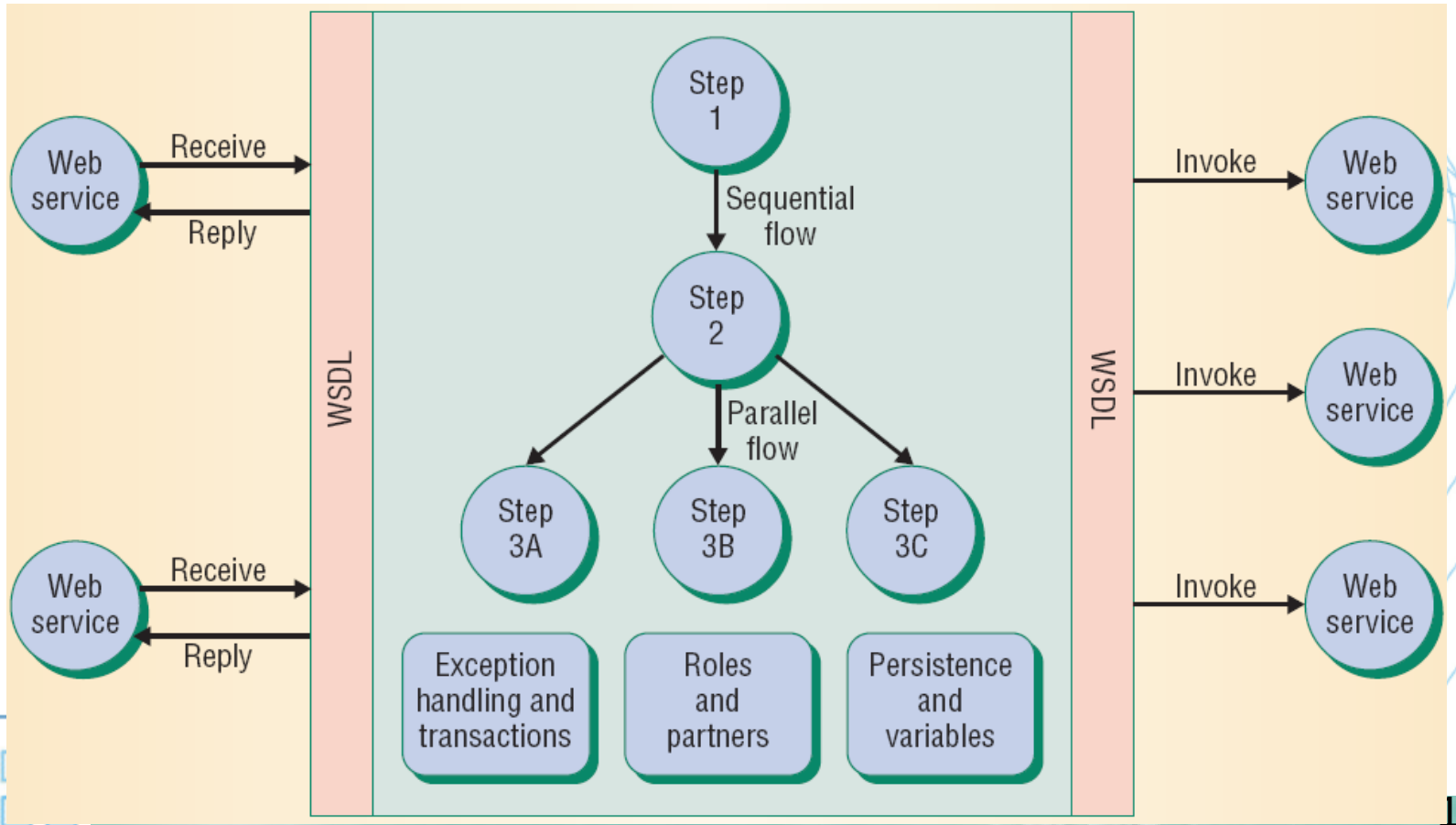
- Choreography的Interaction和Orchestration的receive, invoke
- 由Choreography生成Orchestration模型
  - **Choreography:**
    - SyncDispatchList; GetDispachList; ShowDispatchlist;
  - **Orchestration**
    - A: $Send_{SyncDispatchList}$; $Receive_{GetDispachList}$; $Send_{ShowDispatchlist}$;
    - B: $Receive_{SyncDispatchList}$; $Send_{GetDispachList}$; $Receive_{ShowDispatchlist}$;



Web Services标准规范

服务编制

- Orchestration的本意是“为管弦乐谱曲”：使用五线谱所提供的基本音符，构造一首完整的乐曲。



WS-BPEL语言

BPEL

Business Process Execution Language

业务流程执行语言

- BPEL4WS: Business Process Execution Language for Web Services
- WS-BPEL: Web Services Business Process Execution Language

一种基于XML，用来描述Web服务业务流程的可执行语言

发展历程

- 2002年8月，IBM的WSFL和微软的XLANG两种业务流程语言融合，发布为BPEL4WS 1.0
- 2003年5月，BPEL4WS 1.1 正式发布，并提交给OASIS，改名为WS-BPEL，成立WS-BPEL TC
- 2004年3月，BEA和IBM发布BPELJ白皮书（Java片段扩展）
- 2005年7月，IBM和SAP发布BPEL4People 白皮书（人工任务扩展）
- 2005年9月，IBM和SAP发布BPEL-SPE 白皮书（子流程扩展）
- 2007年4月，WS-BPEL 2.0正式发布，成为OASIS标准， WS-BPEL TC工作结束，关闭
- 2007年6月，BPEL4People 1.0和WS-HumanTask1.0发布，已经提交给OASIS，成立BPEL4People TC。

发展历程

Orchestration in Java

receive
assign
empty
terminate
throw
reply
invoke
switch
scope
while
sequence
组合服务
客户端

```
package org.act.ws.orchestration;

public class BPELProcess {
    public int dataProcess(int a, int b, int c, int d) throws Exception {
        WebService1 ws1 = new WebService1();
        WebService2 ws2 = new WebService2();
        WebService3 ws3 = new WebService3();

        int t1 = ws1.add(a, b);
        int tp = t1;
        int t2 = ws2.multiplex(tp, c);
        int t3 = ws3.minus(t2, d);

        ;

        if(t3 == 0) {
            System.exit(0);
        } else if(t3 < 0) {
            throw new Exception();
        }

        switch (t3) {
        case 50:
            System.out.println("exactly");
        case 80:
            System.out.println("too large");
        default:
            System.out.println("......");
        }

        {
            int i = 0;
            while(i < 10) {
                i++;
                t3 = t3 + 2;
            }
        }
        return t3;
    }

    public static void main(String[] args) {
        BPELProcess pro = new BPELProcess();
        try {
            System.out.println(pro.dataProcess(1,2,8,4));
        } catch(Exception e) {
            System.out.println(e);
        }

    }
}
```

```
package org.act.ws.orchestration;

public class WebService1 {
        public int add(int a, int b) {
                return a + b;
        }
}
```

```
package org.act.ws.orchestration;

public class WebService2 {
        public int multiplex(int a, int b) {
                return a * b;
        }
}
```

```
package org.act.ws.orchestration;

public class WebService3 {
        public int minus(int a , int b) {
                return a - b;
        }
}
```

Orchestration in Java

```
package org.act.ws.orchestration;

public class BPELProcess {
    public int dataProcess(int a, int b, int c, int d) throws Exceptic
        WebService1 ws1 = new WebService1();
        WebService2 ws2 = new WebService2();
        WebService3 ws3 = new WebService3();

        int t1 = ws1.add(a, b);
        int tp = t1;
        int t2 = ws2.multiplex(tp, c);
        int t3 = ws3.minus(t2, d);

        ;

        if(t3 == 0) {
            System.exit(0);
        } else if(t3 < 0) {
            throw new Exception();
        }

        switch (t3) {
        case 50:
            System.out.println("exactly");
        case 80:
            System.out.println("too large");
        default:
            System.out.println("......");
        }

        {
            int i = 0;
            while(i < 10) {
                i++;
                t3 = t3 + 2;
            }
        }
        return t3;
    }

    public static void main(String[] args) {
        BPELProcess pro = new BPELProcess();
        try {
            System.out.println(pro.dataProcess(1,2,8,4));
        } catch(Exception e) {
            System.out.println(e);
        }

    }
}
```

partner link1

partner link2

partner link3

partner link4

WSDL

WSDL

WSDL

WSDL

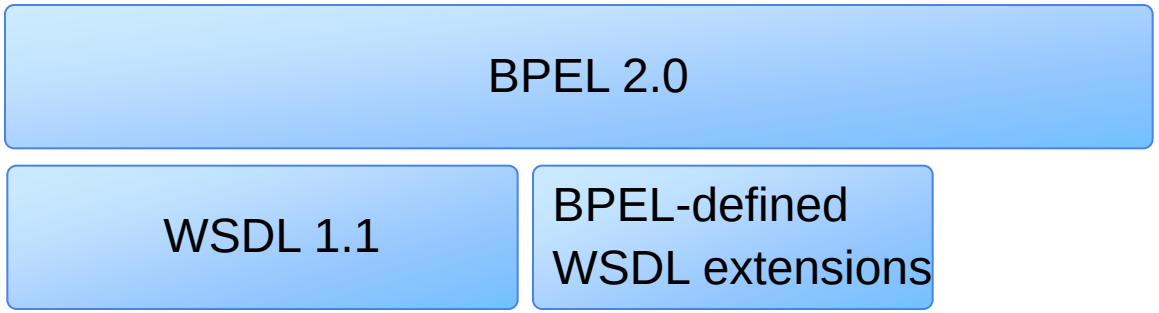
```
package org.act.ws.orchestration;

public class WebService1 {
        public int add(int a, int b) {
            return a + b;
        }
}
```

```
package org.act.ws.orchestration;

public class WebService2 {
        public int multiplex(int a, int b) {
            return a * b;
        }
}
```

```
package org.act.ws.orchestration;

public class WebService3 {
        public int minus(int a , int b) {
            return a - b;
        }
}
```

WS-BPEL与WSDL的关系

- BPEL的作用是将一组现有的服务组合起来，从而定义一个新的Web服务。因此，BPEL基本上是一种实现此种组合的语言。组合服务的接口也被描述为WSDL portType的集合。
- BPEL建立在WSDL服务模型之上，并对其进行了扩展
  - WSDL 定义了允许的操作
  - BPEL定义了 WSDL operations 如何被编排在一起满足特定业务流程
  - BPEL还定义了WSDL的扩展，以支持长周期异步业务流程

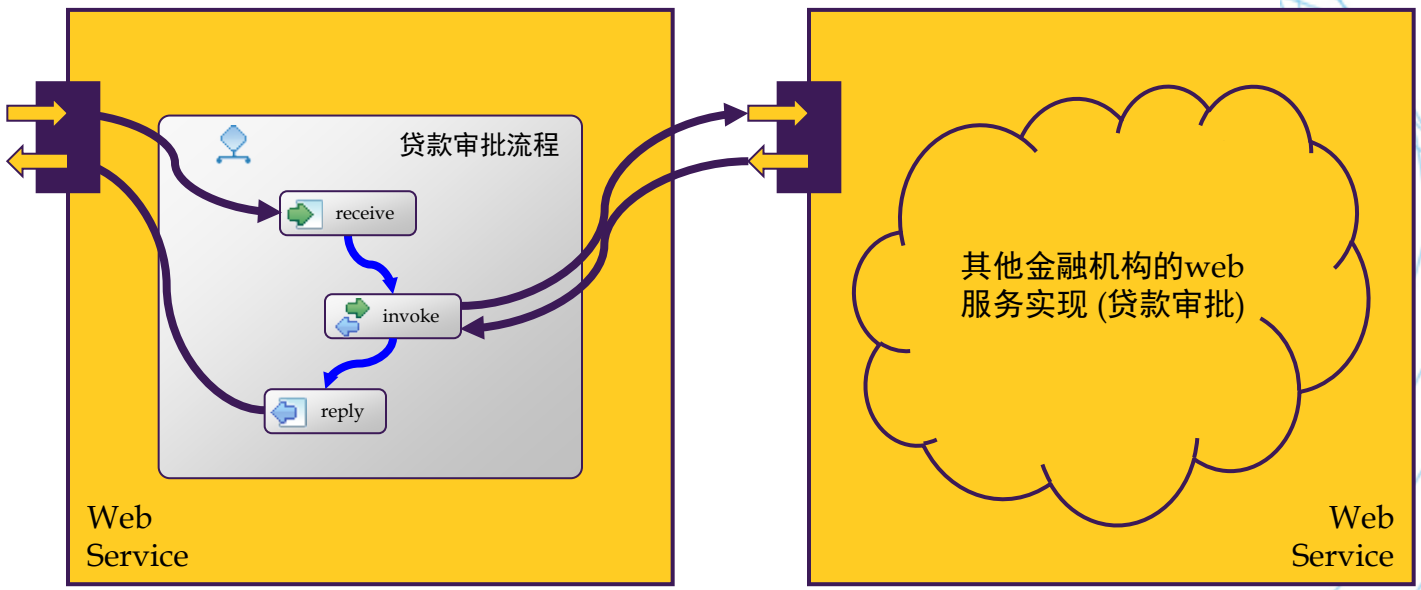


递归的组合模型

BPEL流程以web服务的形式显露给业务伙伴

BPEL流程与业务伙伴提供的web服务交互

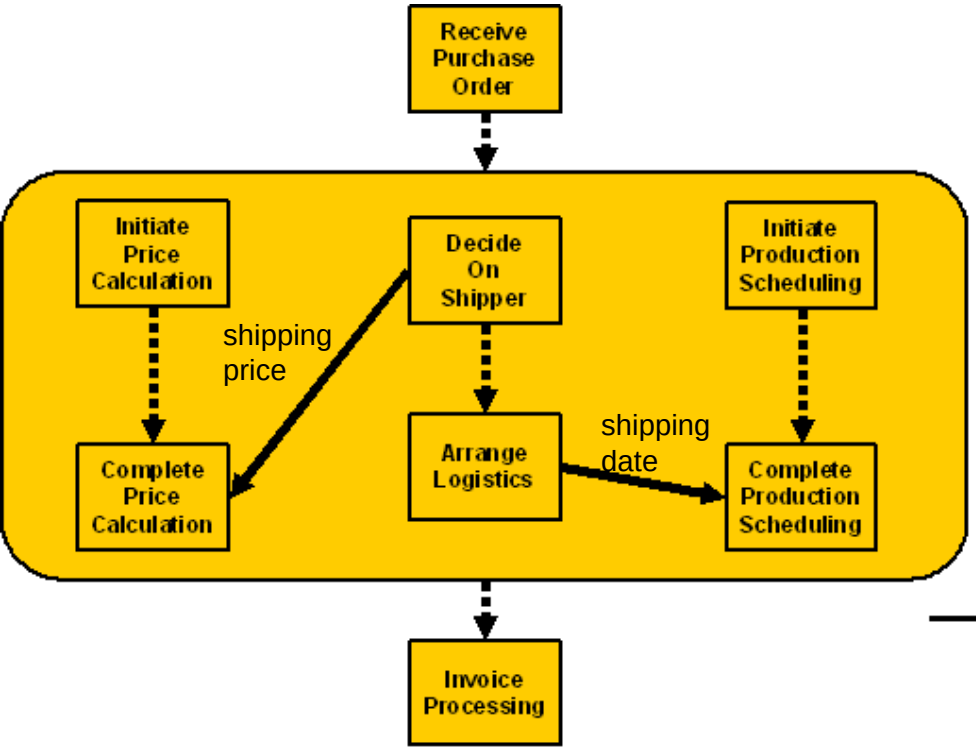
WSDL
Loan
Approval
PortType



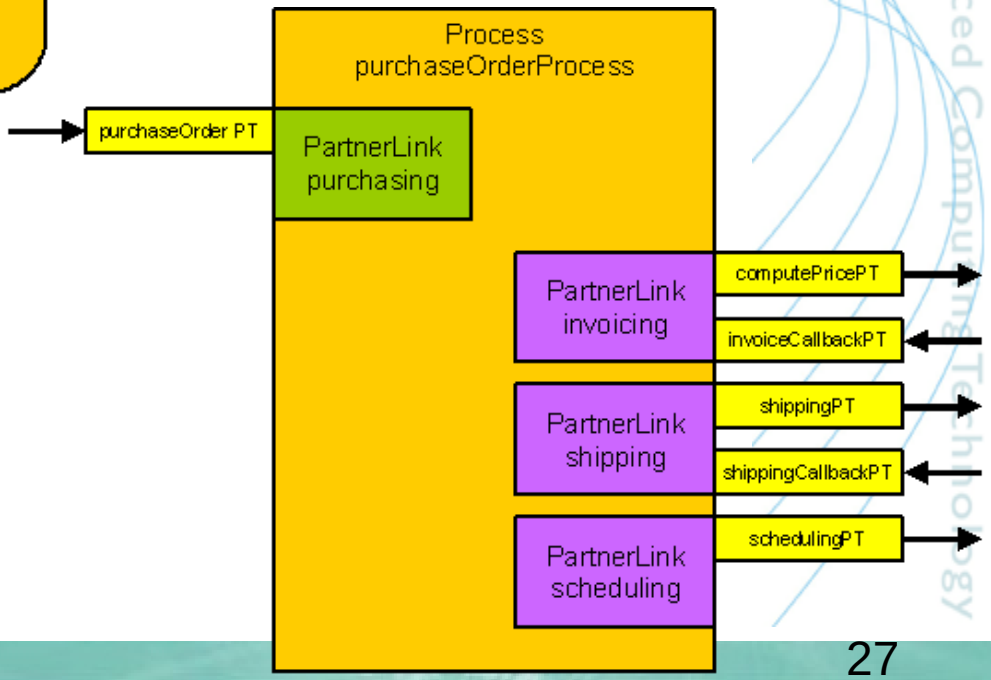
BPEL的基本结构

```
<process name="ncname" targetNamespace="uri"
         queryLanguage="anyURI"?
         expressionLanguage="anyURI"?
         suppressJoinFailure="yes|no"?
         enableInstanceCompensation="yes|no"?
         abstractProcess="yes|no"?>
   <partnerLinks>? ... </partnerLinks>
        <!- 合作伙伴：定义与流程交互的web服务-->
   <variables>? ... </variables>
        <!- 变量：定义流程使用的数据-->
   <correlationSets>? ... </correlationSets>
        <!- 相关集：用于支持异步交互-->
   <faultHandlers>? ... </faultHandlers>
        <!- 故障处理程序：处理异常情况的备选执行路径-->
   <compensationHandlers>? ... </compensationHandlers>
        <!- 补偿处理程序：定义补偿活动-->
   <eventHandlers>? ... </eventHandlers>
        <!- 事件处理程序：特定事件发生发生时执行-->
   activity
        <!- 活动：此处定义流程真正要做的-->
</process>
```

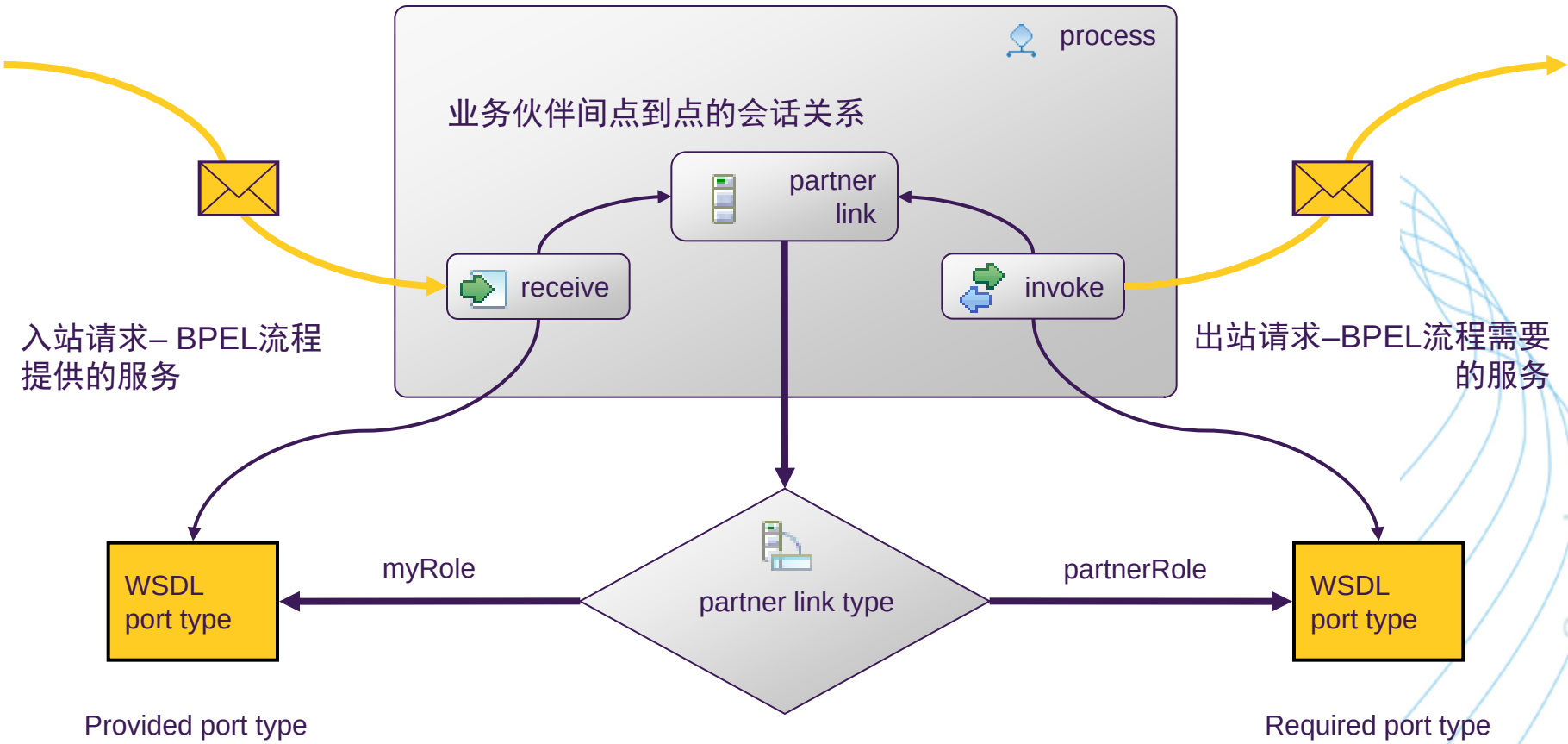
BPEL实例



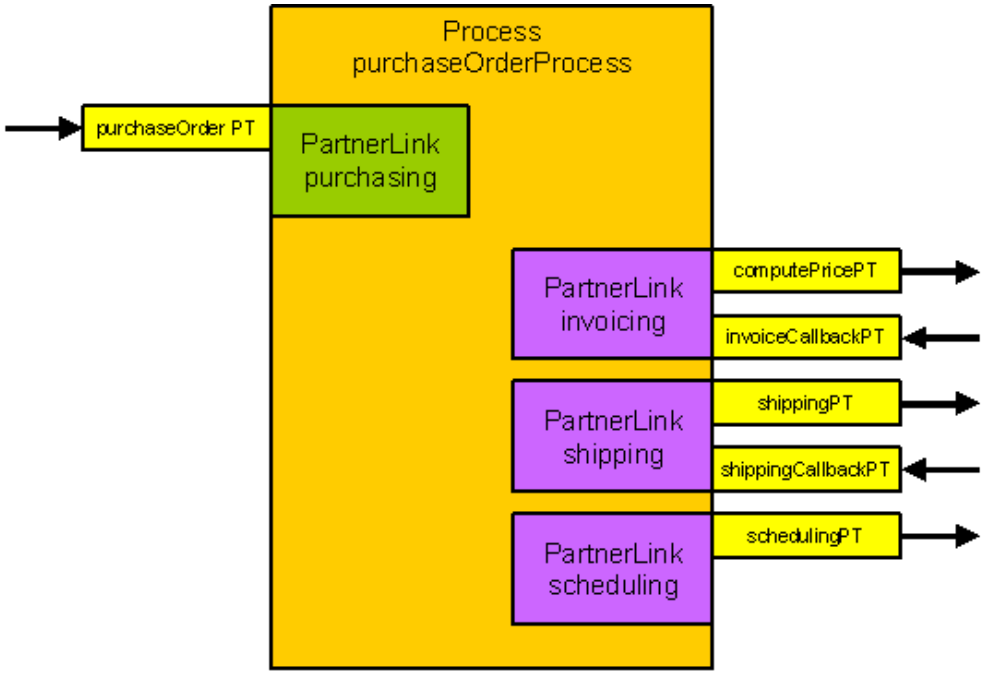
WSDL和BPEL文档见附件



Partner Links



Partner Links



伙伴链接类型PartnerLinkType

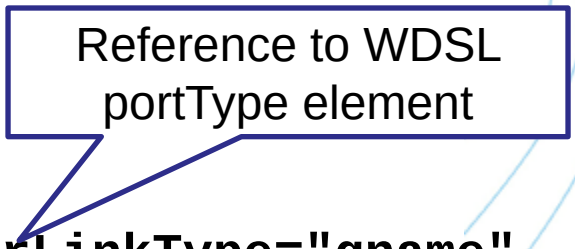
- 为了描述两个服务之间的会话关系，伙伴链接类型定义了会话中每个服务所扮演的“角色”，并且指定了每个服务所提供的portType，以便接收会话的上下文中的消息。
- 伙伴链接类型定义文档可以是独立于任一个服务的WSDL文档的单独构件，也可以被放在定义portType的WSDL文档中，这些portType也被用来定义不同的角色。
- 有些情况下，定义仅包含一个角色的伙伴链接类型是有意义的。在这种伙伴链接情形中，一个服务可以链接任何其他服务。

```
<plnk:partnerLinkType name="ncname">
    <plnk:role name="ncname">
      <plnk:portType name="qname"/>
    </plnk:role>
/plnk:partnerLinkType>
```

每个partnerLintType标签下面有一个或两个role

伙伴链接partnerLinks

- 与业务流程交互的服务被描述成伙伴链接。每个伙伴链由partnerLinkType来描述。
- 属性myRole指出了业务流程本身的角色，而属性partnerRole指出了伙伴的角色。如果partnerLinkType仅有一个角色，那么将根据需要省略其中一个属性。




```
<partnerLinks>
   <partnerLink name="ncname" partnerLinkType="qname"
  myRole="ncname"? partnerRole="ncname"?>+
   </partnerLink>
</partnerLinks>
```

PartnerLink in BPEL

BPEL:

```
<partnerLinks>
 <partnerLink name="customer" partnerLinkType="lns:purchasePLT"
 myRole="purchaseService"/>
</partnerLinks>
```

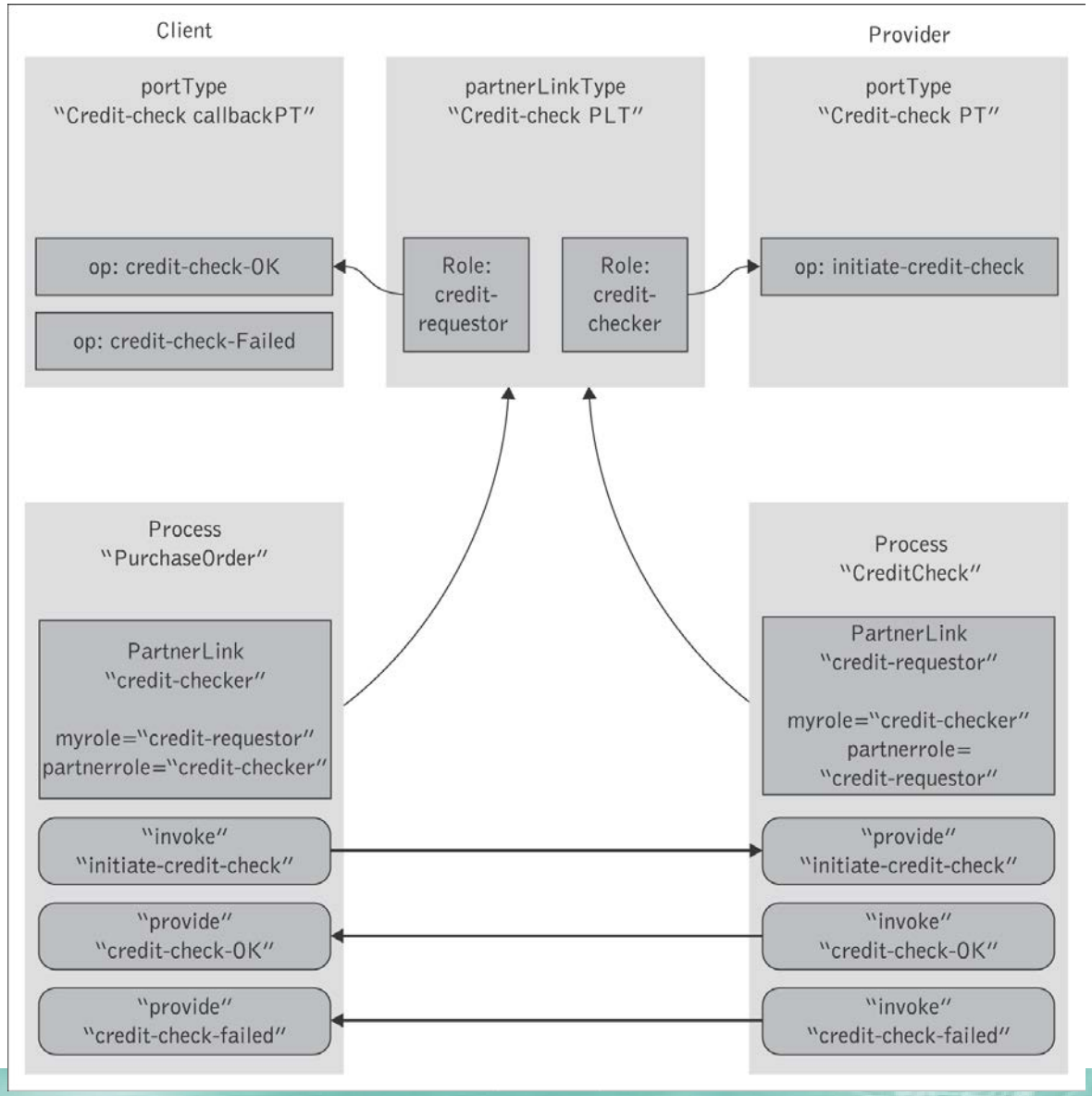
Purchase Process WSDL:

```
<plt:partnerLinkType
name="purchasePLT">
 <plt:role name="purchaseService">
  <plt:portType
name="tns:purchasePT"/>
 </plt:role>
</plt:partnerLinkType>
```

Purchase Process PortType:

```
<portType name="purchasePT">
 <operation
name="sendPurchase">
 </operation>
</portType>
```

PartnerLink与PartnerLinkType



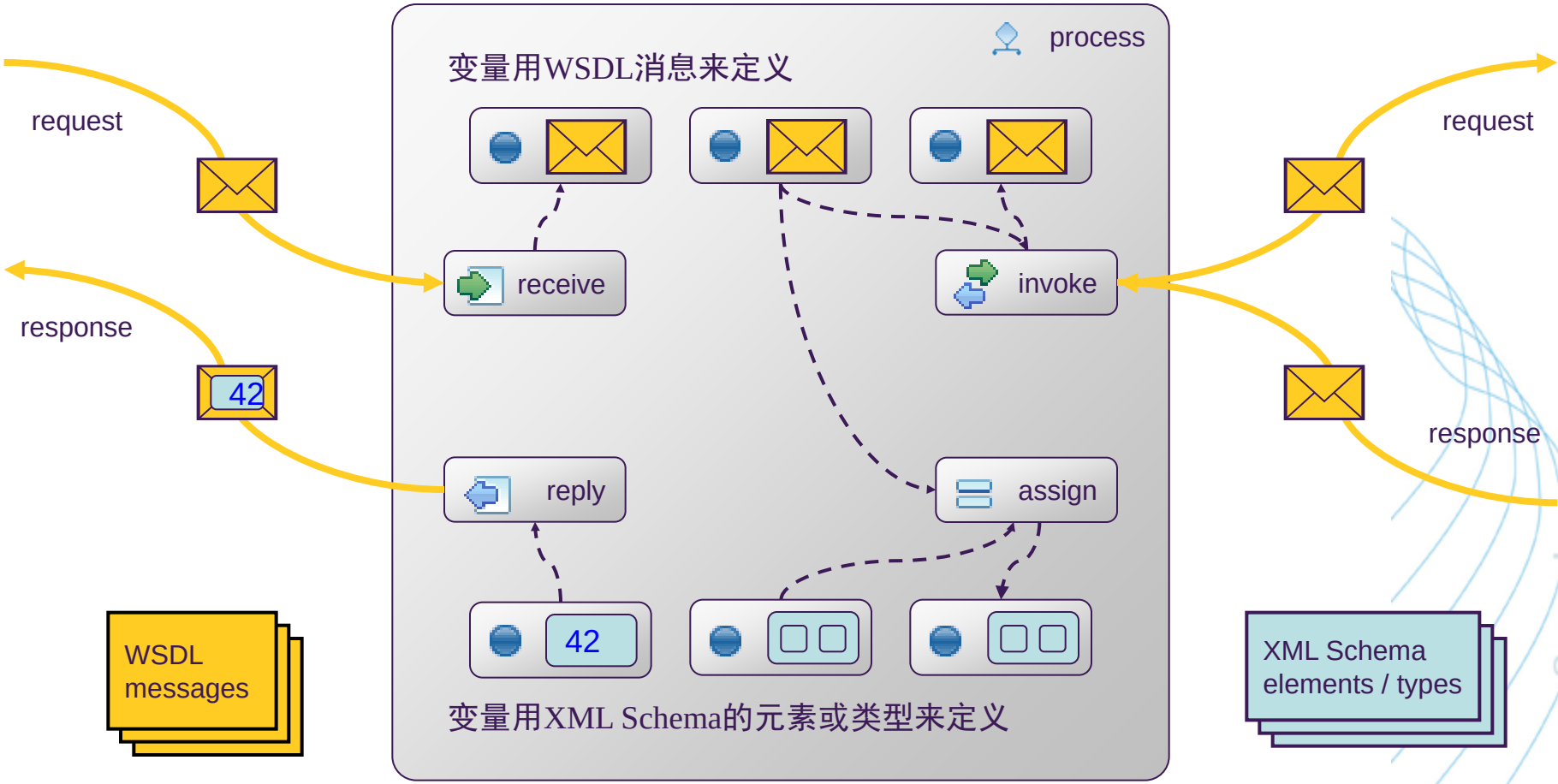
变量Variables

- 业务流程指定了涉及伙伴之间消息交换的有状态交互。业务流程的状态不仅包括被交换的消息，而且还包括用于业务逻辑和构造发送给伙伴的消息的中间数据。
- 每个变量的类型可以是WSDL消息类型、XML Schema简单类型或XML Schema元素。
- 属于全局流程作用域的变量称为全局变量；属于流程作用域的变量称为局部变量；
- 每个变量只有在定义它的作用域和嵌套在它所属于的作用域内的全部作用域中才是可见的

```
<variables>
    <variable name="ncname" messageType="qname"?
              type="qname"? element="qname"?/>+
</variables>
```

```
Java:
int a = obj.method();
System.out.println(obj2.method2(a);
```

变量Variables



BPEL的活动

基本活动

- <receive>
- <reply>
- <invoke>
- <assign>
- <throw>
- <terminate>
- <wait>
- <empty>
- <compensate>

结构化活动

- <sequence>
- <switch>
- <while>
- <pick>
- <flow>
- <scope>

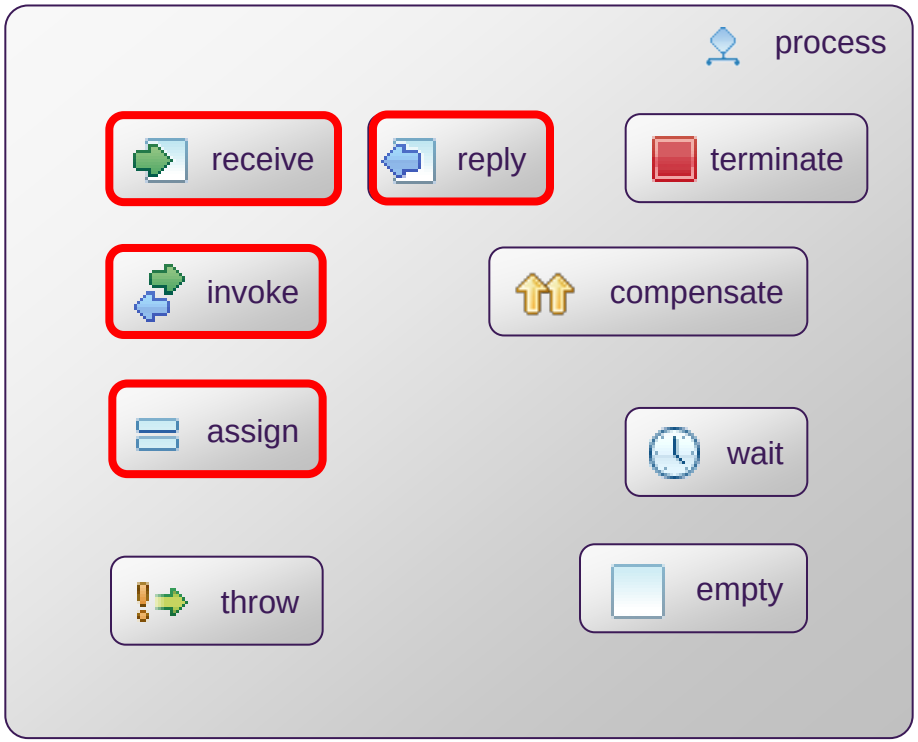
BPEL的基本活动

阻塞并等待匹配的消息来临/发送一个回复消息

调用一个单向或请求-响应操作

用新数据更新变量或partner links的值

从业务流程内部生成一个错误(fault)



立即停止一个业务流程实例的执行

以缺省的顺序在所有完成的子域上调用补偿(compensation)

等待给定的时间段或直到特定时间点

业务流程中的空节点

receive

- <receive>构造业务流程阻塞等待匹配消息的到达
- 实例化业务流程的惟一方法是注解receive活动，把 createInstance属性设置为“yes”。该属性的缺省值是“no”。

```
<receive partnerLink="ncname" portType="qname"
         operation="ncname"
           variable="ncname"? createInstance="yes|no"?
           standard-attributes>
    standard-elements
    <correlations>?
      <correlation set="ncname" initiate="yes|no"?>+
    </correlations>
</receive>
```

reply

- <reply>构造业务流程发送消息以应答通过<receive>接收到的消息。
- receive和reply的组合为流程构成了在WSDL portType上的请求-响应操作。

```
<reply partnerLink="ncname" portType="qname"
        operation="ncname"
        variable="ncname"? faultName="qname"?
        standard-attributes>
    standard-elements
    <correlations>?
       <correlation set="ncname" initiate="yes|no"?>+
    </correlations>
</reply>
```

invoke

- <invoke>构造允许业务流程调用由合作伙伴在portType上提供的单向或请求-响应操作。
- 异步调用仅需要操作的输入变量；同步调用既需要输入变量，又需要输出变量。

```
<invoke partnerLink="ncname" portType="qname"
        operation="ncname"
        inputVariable="ncname"?
        outputVariable="ncname"?
          standard-attributes>
    standard-elements
    <correlations>?
       <correlation set="ncname" initiate="yes|no"? pattern="in|out|out-
   in"/>+
    </correlations>
    <catch faultName="qname" faultVariable="ncname"?>*
      activity
    </catch>
    <catchAll>?
      activity
    </catchAll>
    <compensationHandler>?
      activity
    </compensationHandler>
</invoke>
```

assign

- <assign>构造的作用是用新的数据来更新变量的值。
- assign可以包括任意数量的基本赋值。
- assign还可把端点引用复制到合作伙伴链接，或把合作伙伴链接复制到端点引用（服务的动态绑定）。

```
<assign standard-attributes>
    standard-elements
    <copy>+
      from-spec
      to-spec
    </copy>
</assign>
```

assign

- from-spec必须是以下形式中的一种：
  - <from variable="ncname" part="ncname"?/>
  - <from partnerLink="ncname" endpointReference="myRole|partnerRole"/>
  - <from variable="ncname" property="qname"/>
  - <from expression="general-expr"/>
  - <from> ... literal value ... </from>
- to-spec必须是以下形式中的一种：
  - <to variable="ncname" part="ncname"?/>
  - <to partnerLink="ncname"/>
  - <to variable="ncname" property="qname"/>

throw

- <throw>构造从业务流程中生成故障。
- 使用throw发出内部故障。每个故障需要有一个全局惟一的QName，还可选提供数据的变量。故障处理程序可以使用这种数据，来分析和处理该故障并植入需被发送到其他服务的所有故障消息。

```
<throw faultName="qname"
  faultVariable="ncname"? standard-
  attributes>
    standard-elements
</throw>
```

terminate

- <terminate>可以用于立即终止该terminate活动在其中运行的业务流程实例。
- 所有当前正在运行的活动必须尽可能快地终止，而没有任何故障处理或补偿行为。

```
<terminate standard-attributes>
    standard-elements
</terminate>
```

wait

- <wait>构造允许等待一段给定的时间或等到某一时刻。
- 必须确切地指定wait中一个到期条件。

```
<wait (for="duration-expr" | until="deadline-expr")
   standard-attributes>
     standard-elements
</wait>
```

empty

- <empty>构造允许在业务流程中插入“no-op”指令
- empty可用于需要捕获一个fault，但又不作处理；并行活动之间的同步点。

```
<empty standard-attributes>
    standard-elements
</empty>
```

compensate

- <compensate>构造已正常完成执行的内层作用域上调用补偿。
- compensate命名了执行补偿所在的作用域。仅当作用域正常完成执行之后该作用域的补偿处理程序才可被调用。
- 显式地执行compensate活动的能力是BPEL的应用程序控制的错误处理框架的基础所在。该活动只能用于业务流程的以下部分中：
  - 在作用域的 fault 处理程序中，该作用域直接包括补偿将被执行的作用域
  - 在作用域的补偿处理程序中，该作用域直接包括补偿将被执行的作用域
  - 如果按名称显式地补偿的作用域在循环中被执行，那么在后续的迭代中补偿处理程序的实例将按相反的顺序执行。

```
<compensate scope="ncname"? standard-attributes>
    standard-elements
</compensate>
```

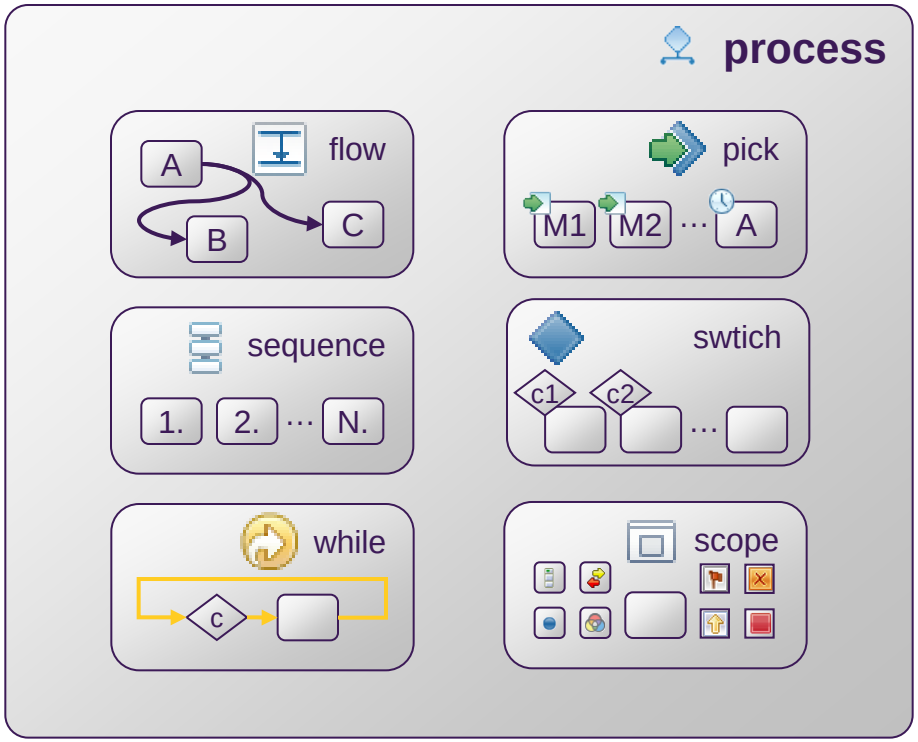
```
<scope name="ReserveAirline">

    <faultHandlers>
    <!-- catch the rollback fault and call -->
    <catch faultName="client:rollbackFault">
            <compensate
            name="CompensateBookings"
            scope="ReserveAirline"/>
    </catch>
    </faultHandlers>

    <!-- define a compensation handler -->
    <compensationHandler>
    <!-- call cancel ticked -->
            <invoke name="Invoke_CancelTicket"
            partnerLink="AirlineBookingService"
            portType="airline:airlineBooking"
            operation="cancelSeat"/>
    </compensationHandler>

    <sequence name="Sequence_1">
            <invoke name="Invoke_BookTicket"
            partnerLink="AirlineBookingService"
            portType="airline:airlineBooking"
            operation="reserveSeat"/>
    </sequence>

            <scope name="ReserveHotel">
            <faultHandlers>
                    <catch
                    faultName="hotel:NoRoomAvailfault"
                    <throw name="RollBack"
                    faultName="client:rollBackFault"/>
                    </catch>
            </faultHandlers>

            <sequence name="Sequence_2"
                    <invoke name="Invoke_ReserveHotel"
                    partnerLink="hotel"
                    portType="ReservationSystem"
                    operation="bookRoom"/>
            </sequence>
            </scope>

</scope>
```

BPEL的结构化活动

包含的活动并发执行，可以用link指定顺序

包含的活动顺序执行

当预定的条件满足时，包含的活动重复执行



阻塞并等待一个消息的来临（或时间超时）

多选一

将一组活动关联起来，使之具有共同的本地变量、伙伴连接、处理器等

BPEL的结构化活动

- 结构化的活动规定了一组活动发生的顺序，描述了创建业务流程的基本活动组成的结构，这些结构表达了涉及业务协议的流程实例间的控制形式、数据流程、故障和外部事件的处理以及消息交换的协调。
- BPEL的结构化活动包括：
  - 顺序控制由sequence、switch和while组成；
  - 活动之间的并发和同步由flow组成；
  - 基于外部事件的不确定的选择由pick组成。
- 递归地使用结构化的活动。

sequence

- <sequence>构造定义一组按顺序先后执行的活动
- 执行顺序是sequence元素中被列出活动的先后顺序
- 当sequence中的最后一个活动完成后，该sequence活动也就完成了

```
<sequence standard-attributes>
    standard-elements
    activity+
</sequence>
```

switch

- <switch>构造允许从一组分支中只选择一个活动分支。
- switch由case元素定义的一个或多个条件分支的有序列表组成，后面可跟也可以不跟一个otherwise分支。
- 以case分支的出现顺序检查，第一个条件是true的分支被选择并被作为被执行的活动。如果有条件的分支都未被选择，那么otherwise分支将被选择。

```
<switch standard-
   attributes>
     standard-elements
     <case
   condition="bool-
   expr">+
        activity
     </case>
     <otherwise>?
        activity
     </otherwise>
</switch>
```

while

<while>构造允许指定反复执行一个活动，直到某个成功条件被满足为止。

```
<while condition="bool-expr" standard-
  attributes>
      standard-elements
      activity
</while>
```

pick

- <pick>构造允许阻塞并等待某一个合适的消息的到达或超时警报响起。当其中一个触发器触发后，相关的活动就被执行，pick也随即完成了。
- pick活动等待一组相互排斥事件中的一个事件的发生，然后执行与发生的事件相关联的活动。
- 如果多个事件发生，那么按照时间发生先后或选择原则确定发生事件。
- 当业务流程的实例的创建是由于接收到一组可能的消息中的一个消息而发生的时，可以使用pick的特殊形式。
- 每个pick活动必须至少包括一个onMessage事件。onMessage事件的语义等同于有关变量属性的可选类型的receive活动。

Pick活动与Receive活动的区别？？

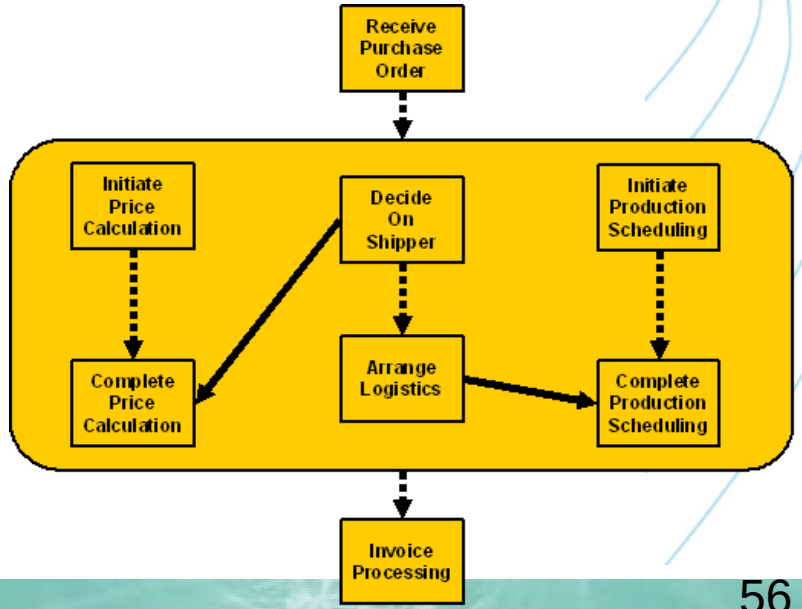
pick

```
<pick createInstance="yes|no"? standard-attributes>
    standard-elements
    <onMessage partnerLink="ncname" portType="qname"
               operation="ncname" variable="ncname"?>+
      <correlations>?
         <correlation set="ncname" initiate="yes|no"?>+
      </correlations>
      activity
    </onMessage>
    <onAlarm (for="duration-expr" | until="deadline-expr")>*
      activity
    </onAlarm>
</pick>
```

flow

- <flow>构造指定一个或多个并行地执行的活动。为了定义任意的控制结构，可以在并行的活动中使用链接。
- flow能进一步表达直接或间接嵌套在其中的活动之间的同步相关性,link构造用来表达这种同步相关性。
- 一个link有一个名称，flow活动的所有链必须在 flow活动中分开定义。活动的标准的source和target元素用来链接两个活动。在flow 活动中声明的每个link必须在该flow中恰好有一个活动作为它的源，恰好有一个活动作为它的目标。

```
<flow standard-attributes>
    standard-elements
    <links>?
      <link name="ncname">+
    </links>
    activity+
</flow>
```



scope

- <scope>构造允许定义嵌套活动，这个嵌套活动有和自己关联的变量、故障处理程序和补偿处理程序。
- 每个scope有一个定义它的正常行为的主要活动。该主要活动可以是一个复杂的结构化的活动，其中有任意深度的许多嵌套的活动。所有的嵌套的活动都共享该scope。

```
<scope variableAccessSerializable="yes|no" standard-
   attributes>
     standard-elements
     <variables> ... </variables>
     <correlationSets> ... </correlationSets>
     <faultHandlers> ... </faultHandlers>
     <compensationHandler> ... </compensationHandler>
     <eventHandlers> ... </eventHandlers>
     activity
</scope>
```

Orchestration in Java

```
package org.act.ws.orchestration;

public class BPELProcess {
    public int dataProcess(int a, int b, int c, int d) throws Exception {
        WebService1 ws1 = new WebService1();
        WebService2 ws2 = new WebService2();
        WebService3 ws3 = new WebService3();

        int t1 = ws1.add(a, b);
        int tp = t1;
        int t2 = ws2.multiplex(tp, c);
        int t3 = ws3.minus(t2, d);

        ;

        if(t3 == 0) {
            System.exit(0);
        } else if(t3 < 0) {
            throw new Exception();
        }

        switch (t3) {
        case 50:
            System.out.println("exactly");
        case 80:
            System.out.println("too large");
        default:
            System.out.println("......");
        }

        {
            int i = 0;
            while(i < 10) {
                i++;
                t3 = t3 + 2;
            }
        }
        return t3;
    }

    public static void main(String[] args) {
        BPELProcess pro = new BPELProcess();
        try {
            System.out.println(pro.dataProcess(1,2,8,4));
        } catch(Exception e) {
            System.out.println(e);
        }

    }
}
```

receive

assign

empty

terminate

throw

reply

invoke

switch

scope

while

sequence

组合服务

客户端

```
package org.act.ws.orchestration;

public class WebService1 {
    public int add(int a, int b) {
        return a + b;
    }
}
```

```
package org.act.ws.orchestration;

public class WebService2 {
    public int multiplex(int a, int b) {
        return a * b;
    }
}
```

```
package org.act.ws.orchestration;

public class WebService3 {
    public int minus(int a , int b) {
        return a - b;
    }
}
```

Orchestration in Java

```
package org.act.ws.orchestration;

public class BPELProcess {
    public int dataProcess(int a, int b, int c, int d) throws Exceptic
        WebService1 ws1 = new WebService1();
        WebService2 ws2 = new WebService2();
        WebService3 ws3 = new WebService3();

        int t1 = ws1.add(a, b);
        int tp = t1;
        int t2 = ws2.multiplex(tp, c);
        int t3 = ws3.minus(t2, d);

        ;

        if(t3 == 0) {
            System.exit(0);
        } else if(t3 < 0) {
            throw new Exception();
        }

        switch (t3) {
        case 50:
            System.out.println("exactly");
        case 80:
            System.out.println("too large");
        default:
            System.out.println("......");
        }

        {
            int i = 0;
            while(i < 10) {
                i++;
                t3 = t3 + 2;
            }
        }
        return t3;
    }

    public static void main(String[] args) {
        BPELProcess pro = new BPELProcess();
        try {
            System.out.println(pro.dataProcess(1,2,8,4));
        } catch(Exception e) {
            System.out.println(e);
        }

    }
}
```

partner link1

partner link2

partner link3

partner link4

WSDL

WSDL

WSDL

WSDL

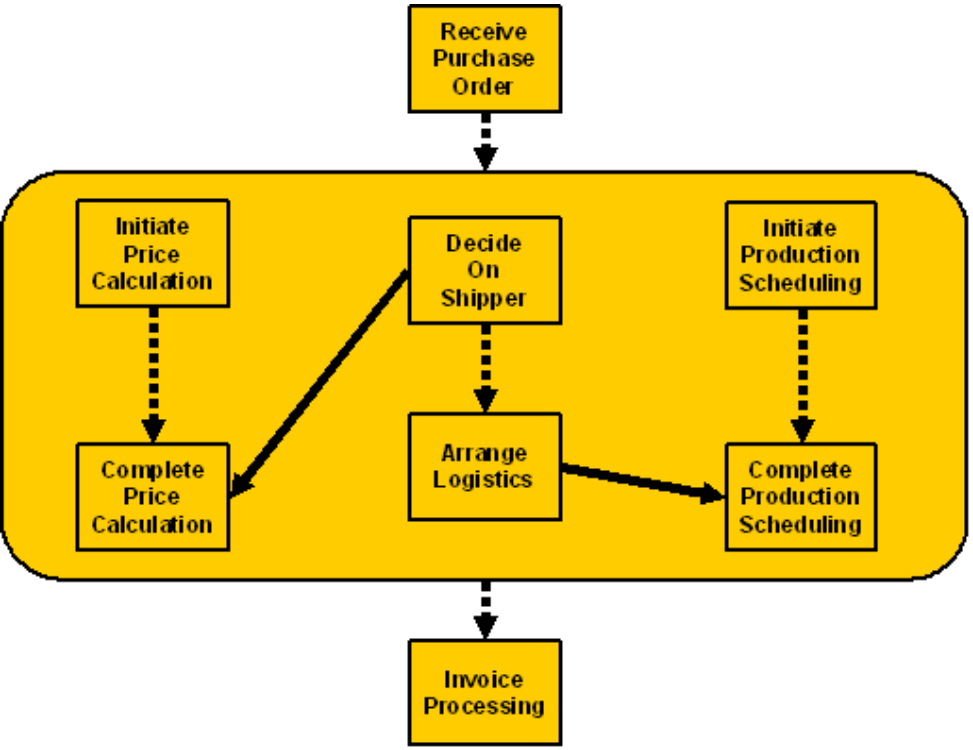
```
package org.act.ws.orchestration;

public class WebService1 {
        public int add(int a, int b) {
            return a + b;
        }
}
```

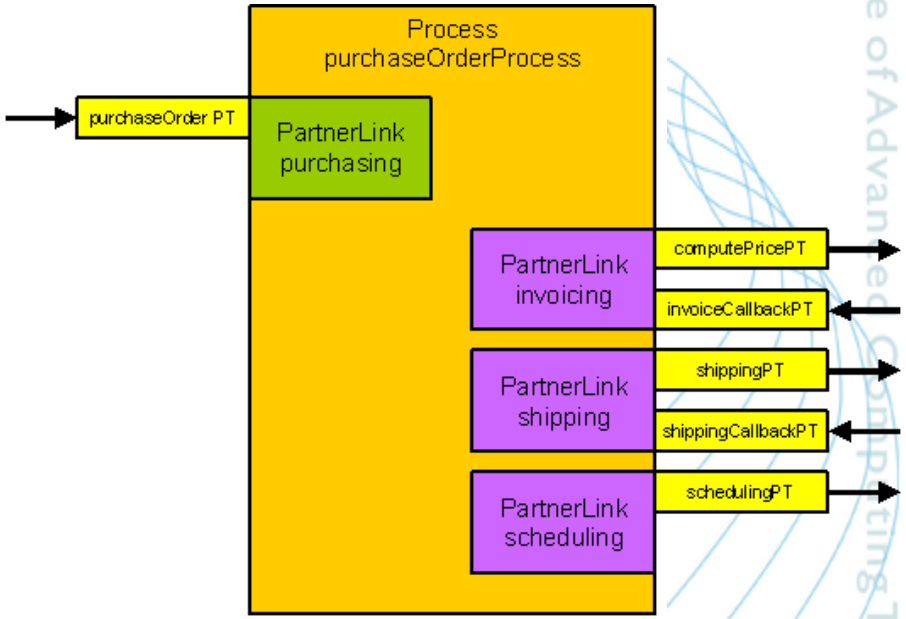
```
package org.act.ws.orchestration;

public class WebService2 {
        public int multiplex(int a, int b) {
            return a * b;
        }
}
```

```
package org.act.ws.orchestration;

public class WebService3 {
        public int minus(int a , int b) {
            return a - b;
        }
}
```

BPEL实例及开发工具

- Eclipse BPEL Editor





服务编排

- W3C Web Services Choreography Working Group
- 编排关注于协作Web服务参与者之间的协同协议
  - Web服务作为对等实体
  - Web服务的交互是长生命周期、有状态的
- Web服务编排实际上是多参与者之间的一个契约，从全局的角度描述了互相协作的Web服务参与者对外的公共可观察行为
- WS-CDL
  - Standardization underway in the W3C Choreography WG

W3C WS-CDL标准组

- 2003年1月成立
- W3C, Oracle, Novell, Enigmatec四个组织的7位成员
- 专家委员会
  - 主要是Pi4技术基金的5位成员
  - 剑桥大学的Robin Milner
- 历史成员包括：Adobe, Sun, Microsoft, SAP, HP, Hitachi等

WS-CDL 历史

- 2003年8月
  - First requirements document for Web services choreography.
- 2004年4月
  - First Working Draft of the Web services choreography specification.
- 2004年12月
  - Last Call Working Draft of the Web services choreography specification.
- 2005年12月
  - Candidate Recommendation for the Web services choreography specification.
- 2006年6月
  - Proposed Recommendation for the Web services choreography specification.
- 2006年12月
  - Working Group ends.

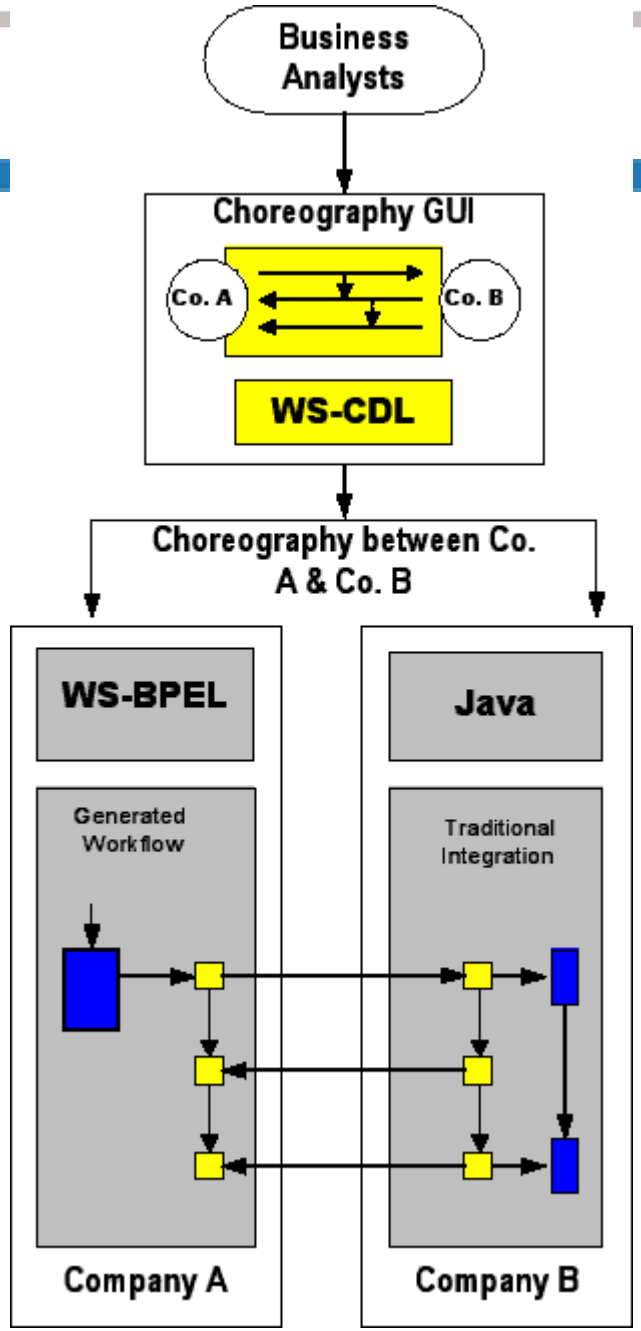
WS-CDL相关产品

- Pi4SOA
  - HatTrick Software的工具
  - Pi4技术基金 (http://www.pi4tech.org/)
- WS-CDL Eclipse
  - http://sourceforge.net/projects/wscdl-eclipse/
- WS-Engineer Plug-in for LTSA
  - 帝国理工学院
  - http://www.doc.ic.ac.uk/ltsa/eclipse/wsengineer/
- WS-CDL+ Execution Engine
  - 东南大学
  - http://wscomposition.seu.edu.cn/index.html
- TrustCom
  - Secure and Trusted Virtual Organization Management
  - UML to CDL
  - CDL to BPEL

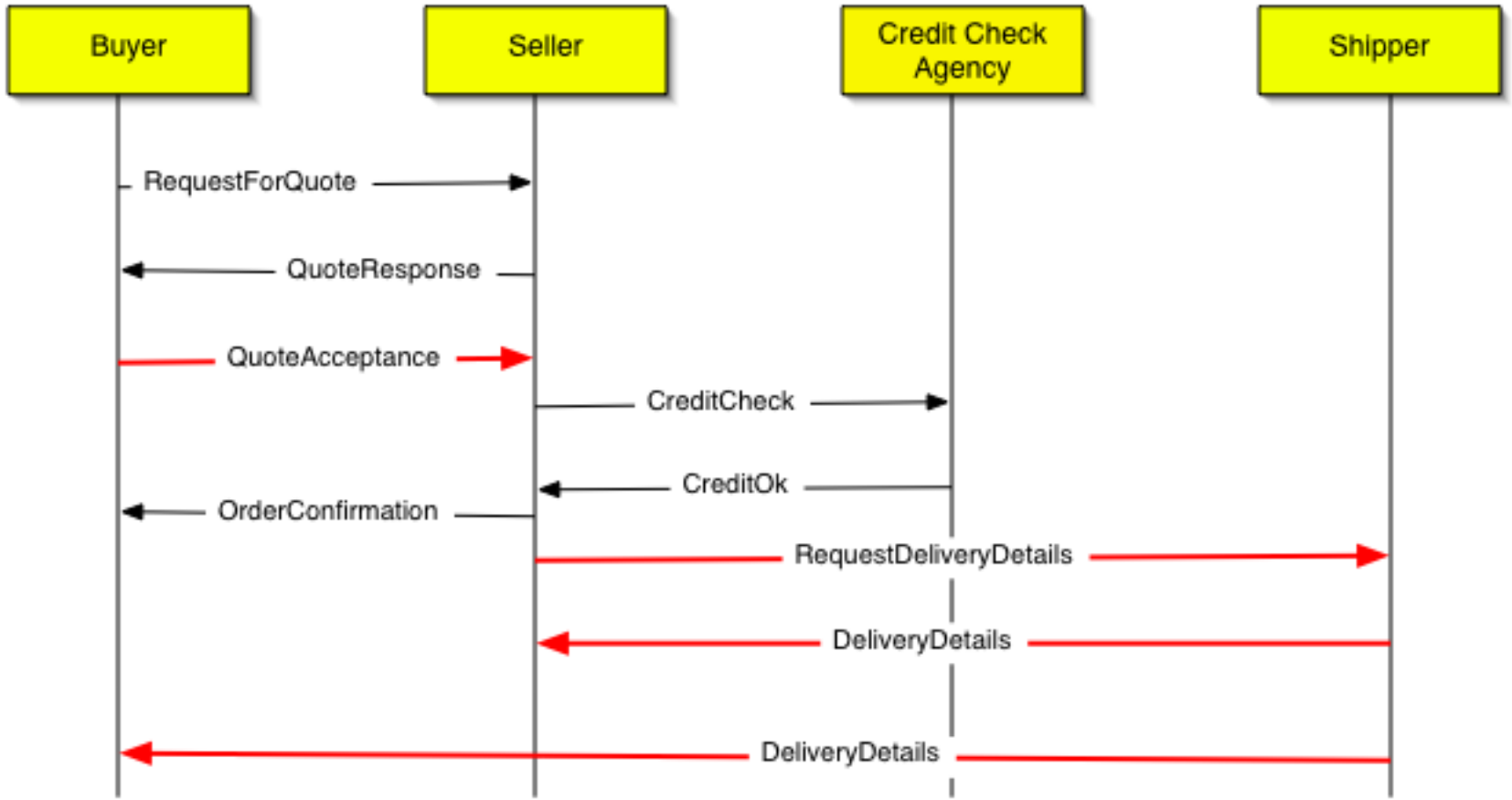
Pi4SOA

- Pi4 技术基金的开源工具
  - Eclipse插件
  - 图形化模型编辑环境
  - 静态类型检查
  - 模型转换

    - **Export**
      - WS-CDL
      - WSDL
      - HTML
      - BPMN
    - **Generate**
      - UML artifacts
      - Java (for J2EE)
      - Java (for Axis)
      - BPEL
    - **Monitor**
      - Against WS-CDL description
      - Legacy and/or generated
- Project members
  - **Steve Ross-Talbot, Gary Brown (lead), Nobuko Yoshida, Kohei Honda, Marco Carbone, Robin Milner, Charlton Barretto**

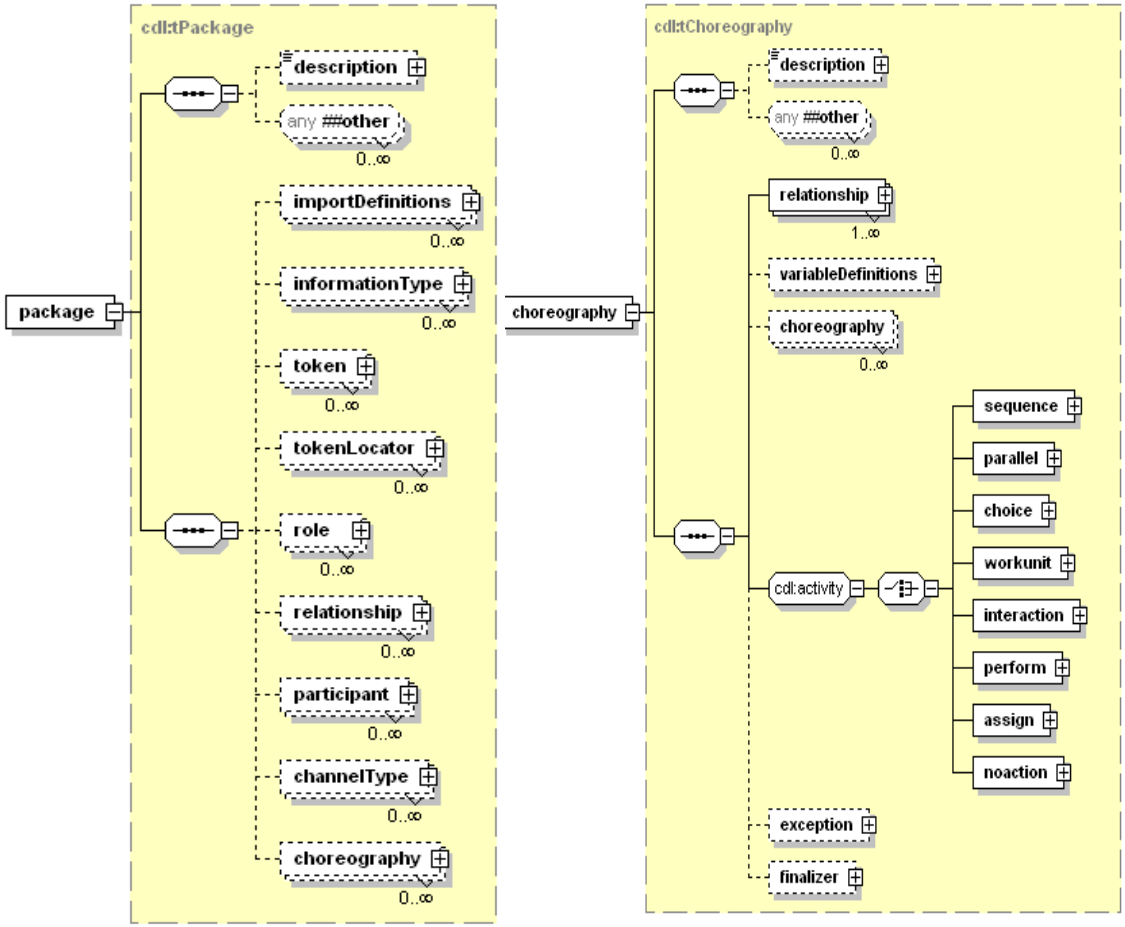


WS-CDL - An example



Normal Collaboration

WS-CDL elements



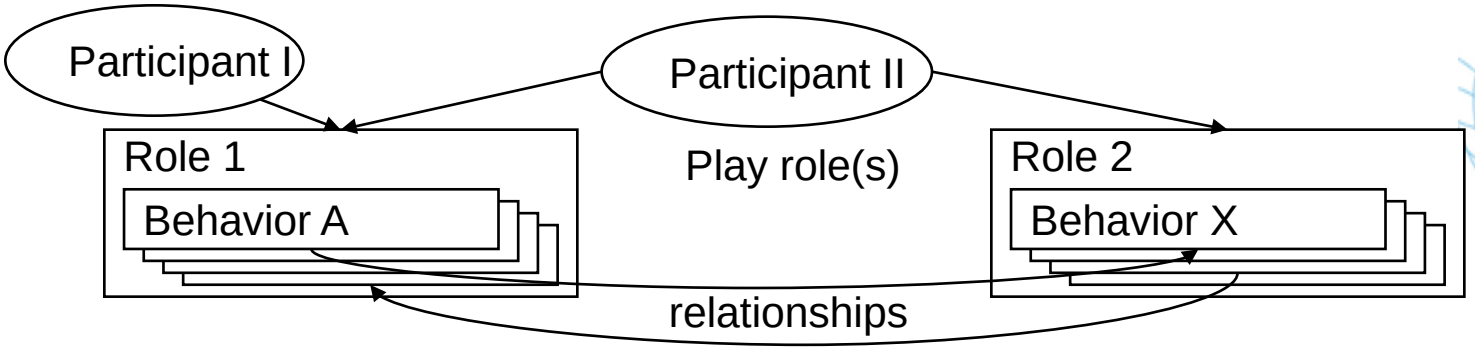
- Package
- Data concepts
- Participants & Roles
- Channels
- Choreography

WS-CDL文档结构

- <package name="NCName" author="xsd:string"? version="xsd:string"? targetNamespace="uri" xmlns="http://www.w3.org/2005/10/cdl">
  - <informationType/>*
  - <token/>*
  - <tokenLocator/>*
  - <roleType/>*
  - <relationshipType/>*
  - <participantType/>*
  - <channelType/>*
  - Choreography-Notation*
- </package>

参与者

- Roles, Participants & Relationships

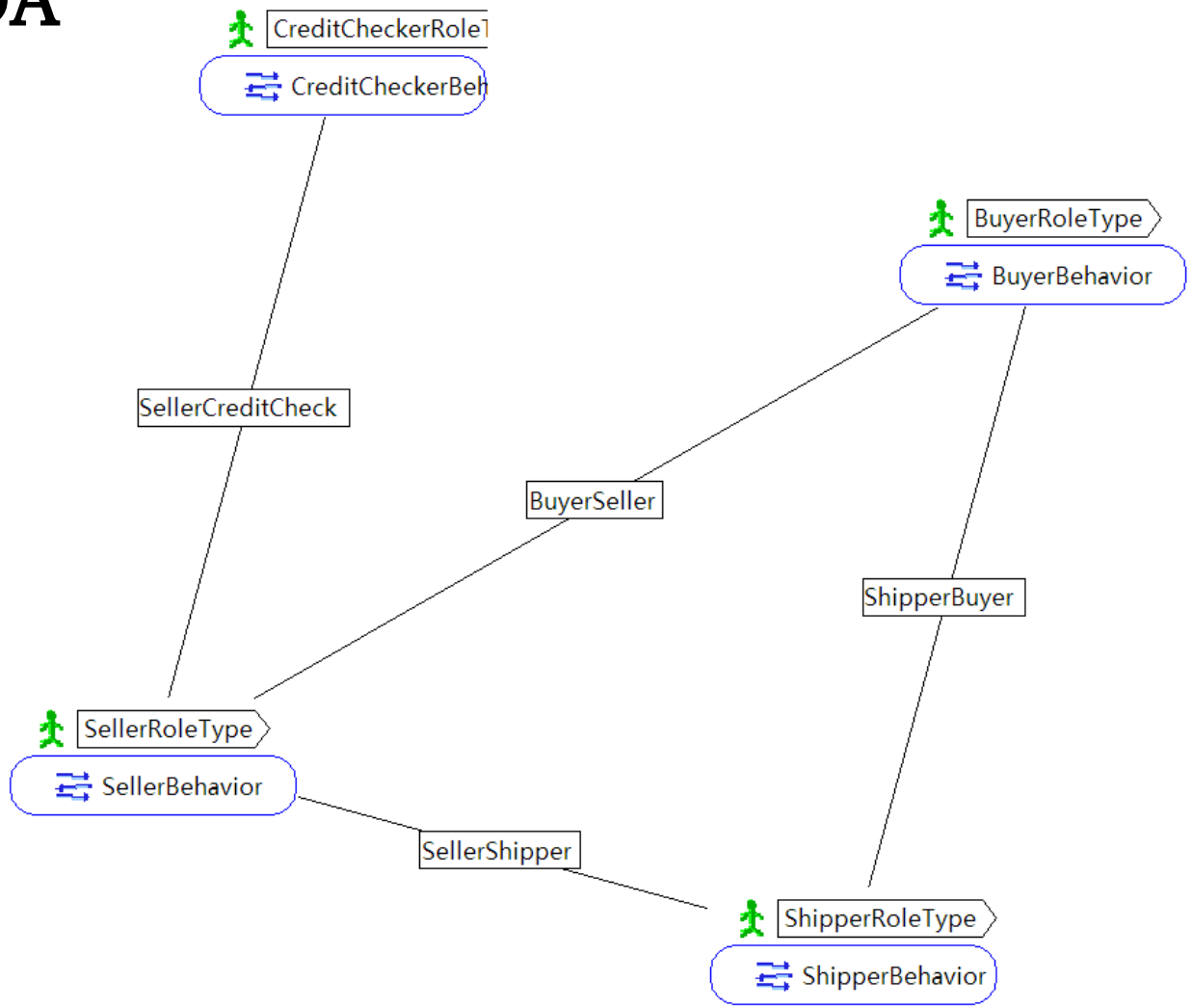


WS-CDL: behavior

- Interactions
- Basic activities
  - noAction
  - silentAction
  - finilize
  - assign
- Structured activities
  - Sequence
  - Parallel
  - Choice
- Composition of choreographies
  - perform
- Workunit

WS-CDL实例及开发工具

- Pi4SOA



WS-BPEL和WS-CDL对比

- 目标不同
- BPEL成熟，有很多国际大公司的产品
- WS-CDL尚未成熟
  - No support from IBM, MS
  - Apparently initially from Oracle
  - 工业界只有Oracle, Adobe等支持
  - 学术界有一些原型系统
- 学术界：服务编制和编排是研究热点
- 未来
  - 正面：WS-CDL是SOA的发展方向
  - 反面：没有企业的支持，没前途

WS-BPEL vs. WS-CDL

	WS-BPEL	WS-CDL
用途	系统实现：可执行流程	系统描述
运行模式	有中心、层次关系	无中心、对等关系
成熟度	正式规范、得到厂商支持、软件工具众多	非正式规范、得到厂商支持少、软件工具少
适用场合	域内小粒度服务组合	域间大粒度服务协作
描述对象	业务流程	消息交互的时序
软件工程方法	自底向上	自顶向下

WS-BPEL & WS-CDL

- WS-CDL ≈ N个目标一致的WS-BPEL

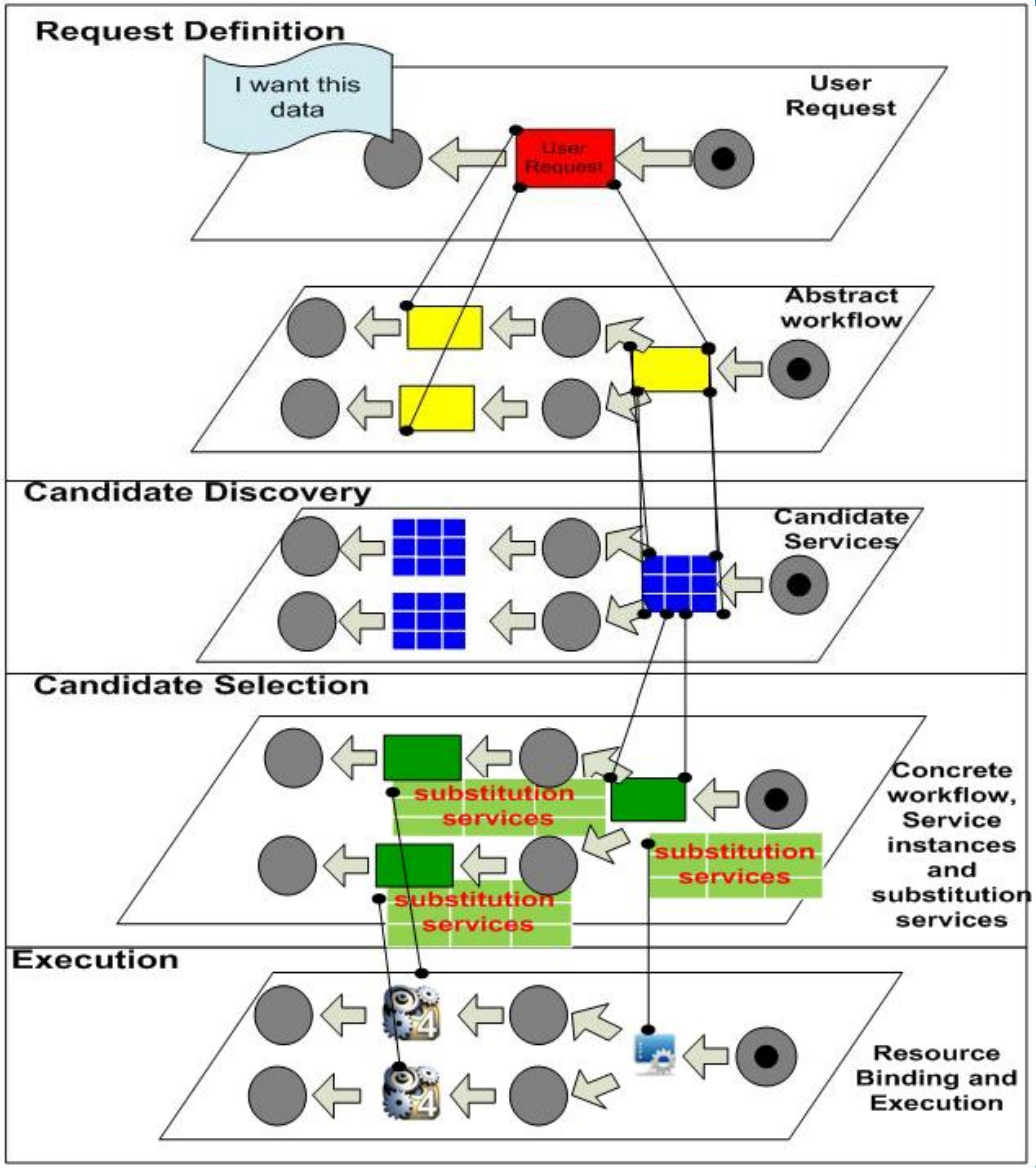
服务组合的类型

- Web服务组合
  - 手动：静态组合
  - 半自动：动态组合
  - 全自动组合
- RESTful服务/Web API组合

动态服务组合

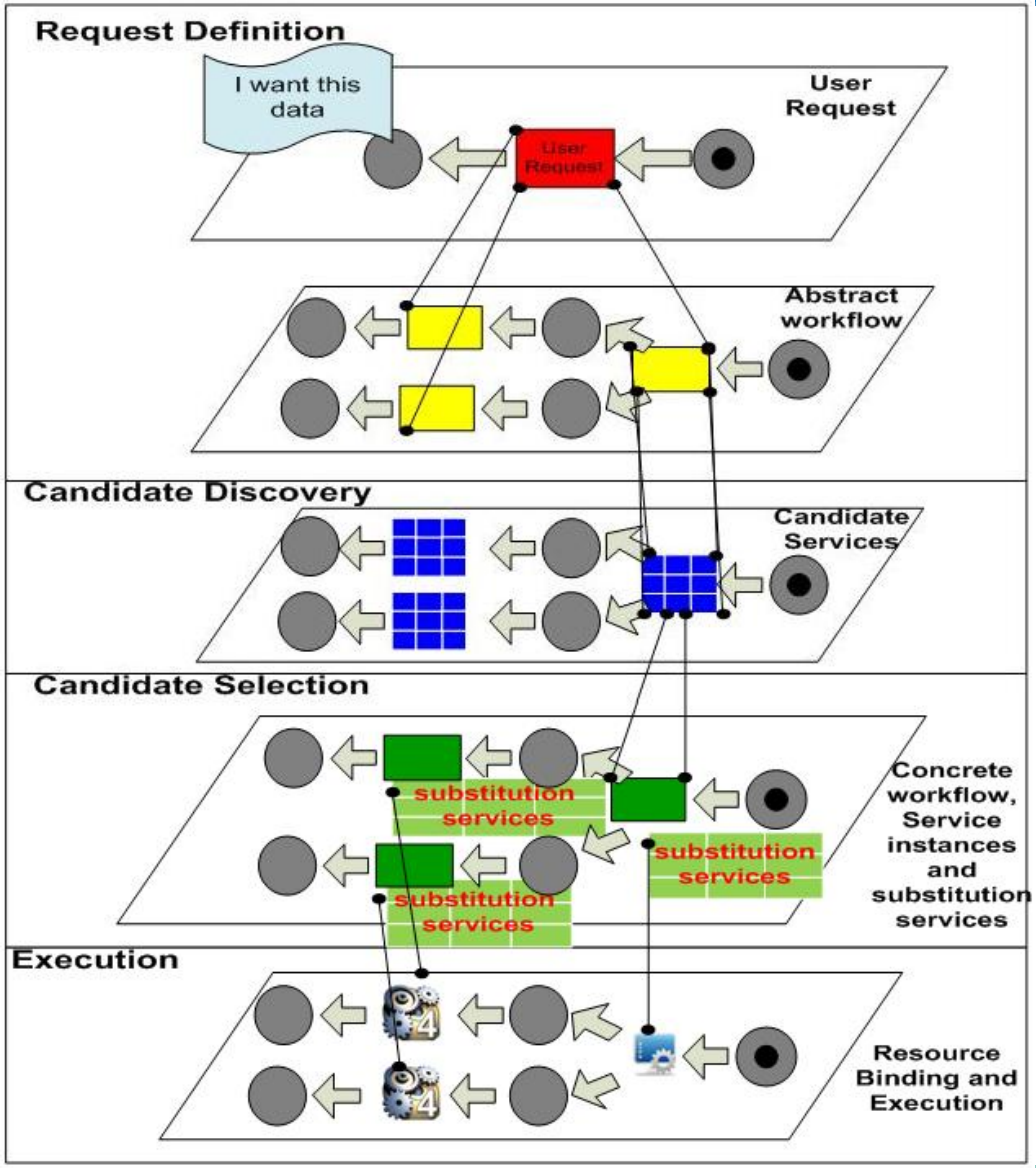
- 根据业务需求，构建业务流程模型
- 每个流程活动初始时并不绑定具体的服务
- 组合服务执行时，根据上下文以及当时的服务状态动态选择服务进行绑定
- 类型
  - 动态语义服务组合
  - 动态QoS服务组合

动态语义服务组合



基于语义的服务发现，已在服务发现章节描述

动态QoS服务组合

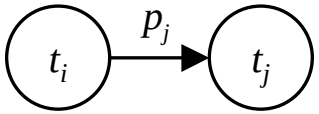


在功能相同的服务里，根据QoS选择服务进行动态绑定。QoS预测的相关内容已在服务推荐介绍。

组合服务QoS的计算-Workflow Reduction

T: time; C: cost;
R: reliability

Reduction of a Sequential System

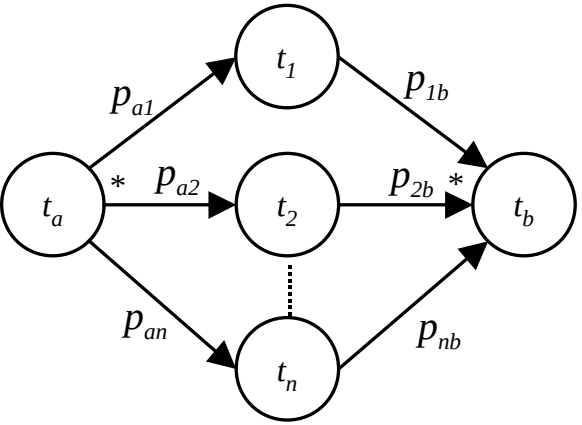


(a)

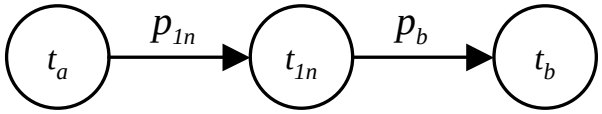
(b)

$$T(t_{ij}) = T(t_i) + T(t_j)$$
$$C(t_{ij}) = C(t_i) + C(t_j)$$
$$R(t_{ij}) = R(t_i) * R(t_j)$$

Reduction of a Parallel System



(a)



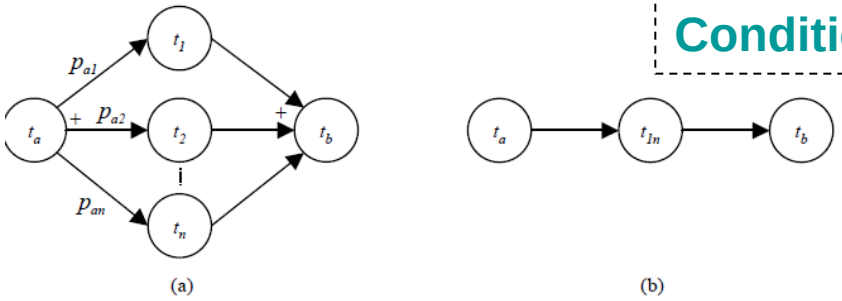
(b)

$$T(t_{1n}) = \mathrm{Max}_{I \in \{1..n\}} \{T(t_i)\}$$
$$C(t_{1n}) = \sum_{1 \le i \le .n} C(t_i)$$
$$R(t_{1n}) = \prod_{1 \le i \le .n} R(t_i)$$

Jorge Cardoso, Amit P. Sheth, John A. Miller, Jonathan Arnold, Krys Kochut: Quality of service for workflows and web service processes. Journal of Web Semantics, 1(3): 281-308 (2004)

组合服务QoS计算-Workflow Reduction（续）

Reduction of a Conditional System



$$T(t_{1n}) = \sum_{1 \le i \le n} p_{ai} * T(t_i)$$

$$C(t_{1n}) = \sum_{1 \le i \le n} p_{ai} * C(t_i)$$

$$R(t_{1n}) = \sum_{1 \le i \le n} p_{ai} * R(t_i)$$

Reduction of a Simple Loop System

$$T(t_{li}) = \frac{T(t_i)}{1 - p_i}$$

$$C(t_{li}) = \frac{C(t_i)}{1 - p_i}$$

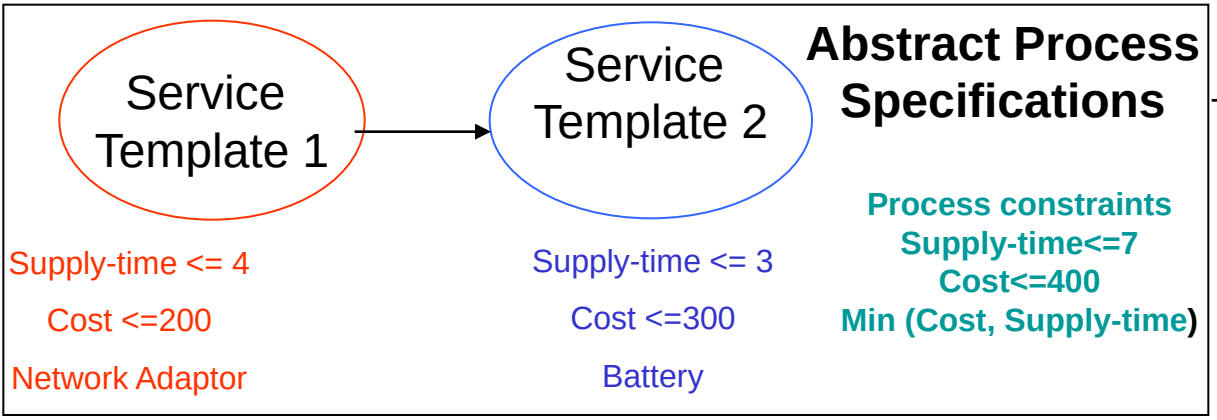
$$R(t_{li}) = \frac{(1 - p_i) * R(t_i)}{1 - p_i R(t_i)}$$

Jorge Cardoso, Amit P. Sheth, John A. Miller, Jonathan Arnold, Krys Kochut: Quality of service for workflows and web service processes. Journal of Web Semantics, 1(3): 281-308 (2004)

假设每个服务有n_i个副本(可用性计算)

组合服务可用度	说明	计算公式
$A_{Sequence}$	顺序组合的可用度	$A_{Sequence}=\prod_{j=1}^{m}(1-\prod_{i=1}^{n_j}(1-A_{s_j^i}));\ i=1,2,...,n_j;\ j=1,2,...,m$
$A_{Parallel}$	并行组合的可用度	$A_{Parallel}=\prod_{j=1}^{m}(1-\prod_{i=1}^{n_j}(1-A_{s_j^i}));\ i=1,2,...,n_j;\ j=1,2,...,m$
A_{Choice}	选择组合的可用度	$A_{Choice}=\sum_{j=1}^{m}(1-\prod_{i=1}^{n_j}(1-A_{s_j^i}))*p_j;\ \sum_{j=1}^{m}p_j=1;\ \begin{matrix}i=1,2,...,n_j;\\ j=1,2,...,m\end{matrix}$
$A_{Iteration}$	循环组合的可用度	$A_{Iteration}=(1-p)*(1-\prod_{i=1}^{n_j}(1-A_{s_j^i}))/(1-p*(1-\prod_{i=1}^{n_j}(1-A_{s_j^i})))$

共有m种服务，每种服务有n_j个副本



Service templates and service constraints

Discovery Engine

ST=4 C=200 | ST=3 C=180 | ST=3 C=200 | ST=1 C=300 | ST=2 C=100 | ST=3 C=250

Domain constraints in ontologies

Domain Reasoner (DL)

Optimizer (ILP)

Objective Function and Process constraints
Min (supply-time + cost)

Most optimal set cannot be chosen because of inter service dependencies
Network Adaptor from supplier 1 does not work
battery from supplier 2

ST=4 C=200 | ST=3 C=180 | ST=4 C=200 | ST=3 C=180

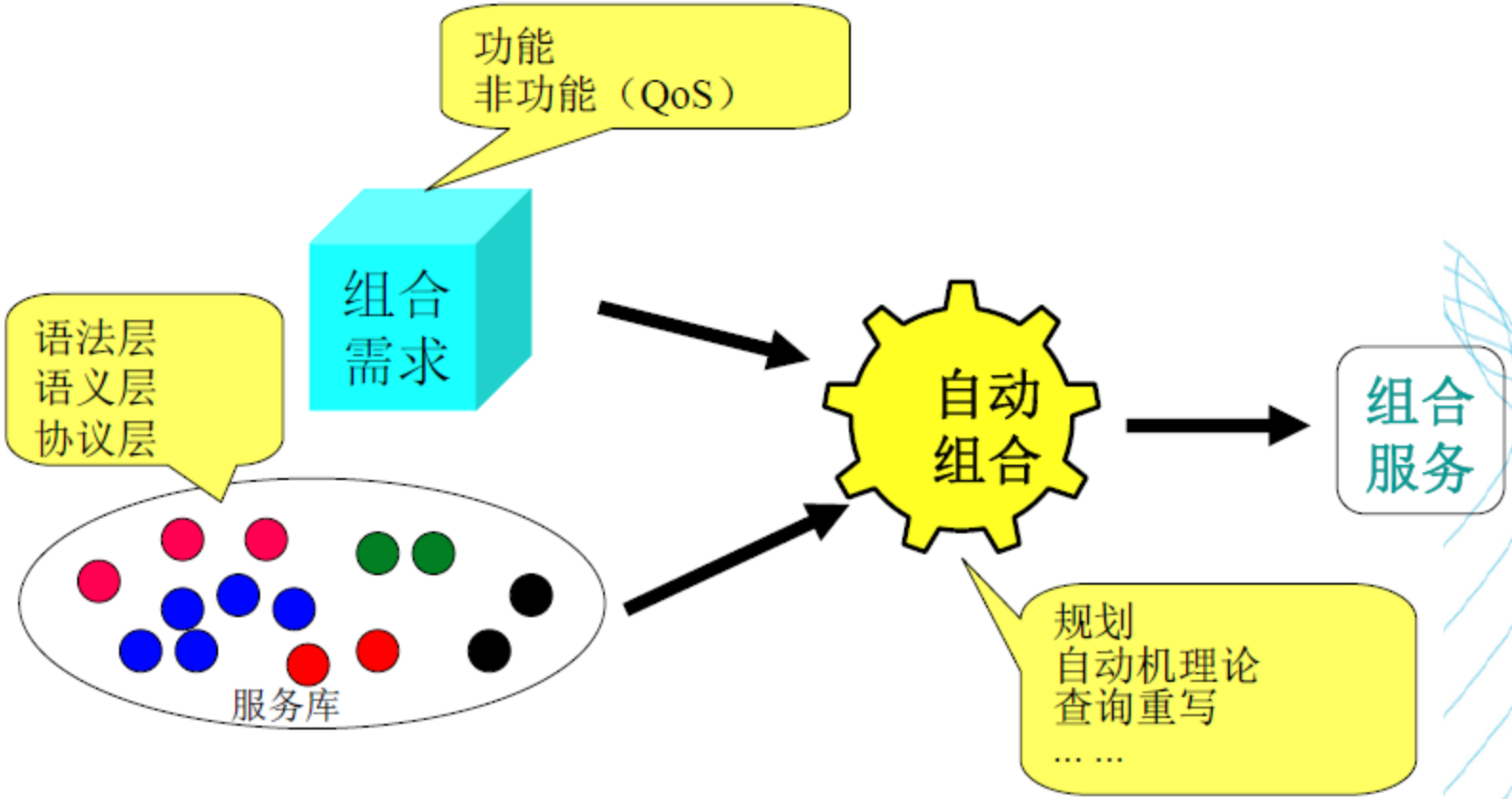
Ranked Set

Rohit Aggarwal, Kunal Verma, John A. Miller and William Milnor, "Constraint Driven Web Service Composition in METEOR-S," Proceedings of the 2004 IEEE International Conference on Services Computing (SCC 2004), Shanghai, China, September 2004

服务组合的类型

- Web服务组合
  - 手动：静态组合
  - 半自动：动态组合
  - 全自动组合
- RESTful服务/Web API组合

自动服务组合研究现状



自动服务组合研究现状(续)

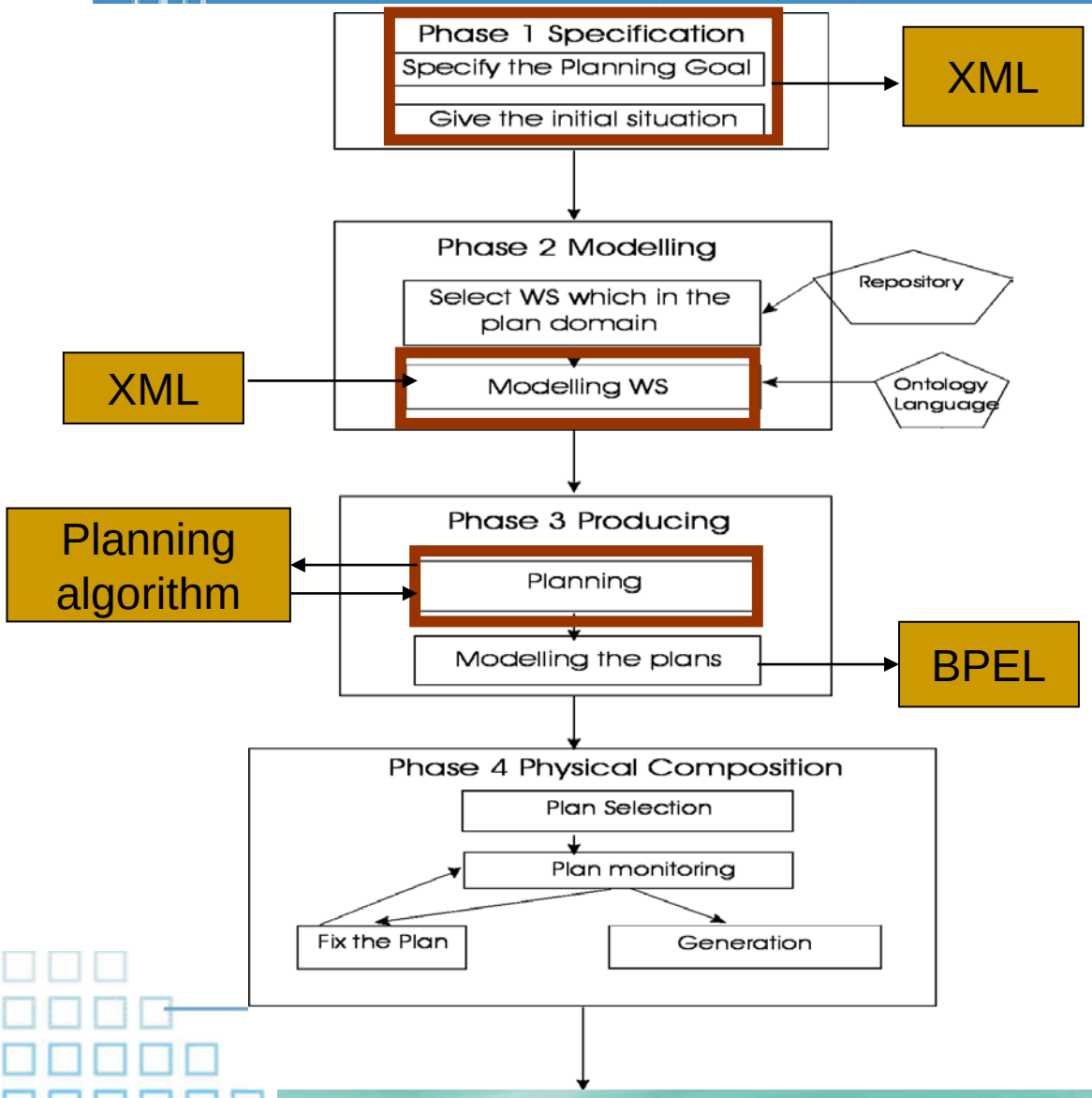
- 语法层自动服务组合
  - 基于WSDL接口描述，把服务建模为输入/输出函数
  - 数据集成、类型推导、查询优化（Active XML）
- 语义层自动服务组合
  - 设计一种计算机够互相理解并能充分表示Web服务的内容、功能、属性、接口以及规则和限制条件的语言，研究Web服务的自动组合
  - Meteor-S 、SWSF 、SWORD 、SHOP2
- 协议层自动服务组合
  - 服务基于其业务协议进行建模，刻画了服务外部可观察的行为
  - Roman模型、 Colombo模型 、MoSCoE、 SWS模型、 ASTRO框架

组合服务的形式化模型

- 自动机模型：**Roman model**
- **Petri网模型: 工作流网**
- 进程代数模型：**process algebra**
- 情境演算: **situation calculus**
- 语义模型: **semantic web**
- 会话模型: **conversation model**
- **CTL模型：**

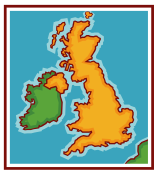
- 混合模型: **hybrid model**
  - **FLOWS/**
  - **Colombo model**

基于规划的自动服务组合



- 1.目标: 领域兴趣，目标，搜索深度约束等
- 初始情境：客户端要求的初始输入消息和状态，初始化条件和参数
- 2. 注册中心：存储服务的domain, ontology type and web addresses
- 根据服务发布的信息进行检索，将服务ontology映射到一个公共的ontology
- 3. 产生一个plan的集合
- 4. 客户端选择最佳plan
- 产生可执行流程(如BPEL)
- 监控流程的执行，若出错可根据第3步中的可选plan自动修复

实例

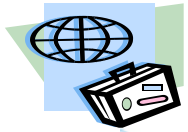
输入：家庭地址和计算机IP地址

目标：购买电视后，得到快递的电视，并获得保险

&& After 

T ∧◊(D∧I)

WS1=Locate IP 

WS2= TV Information 

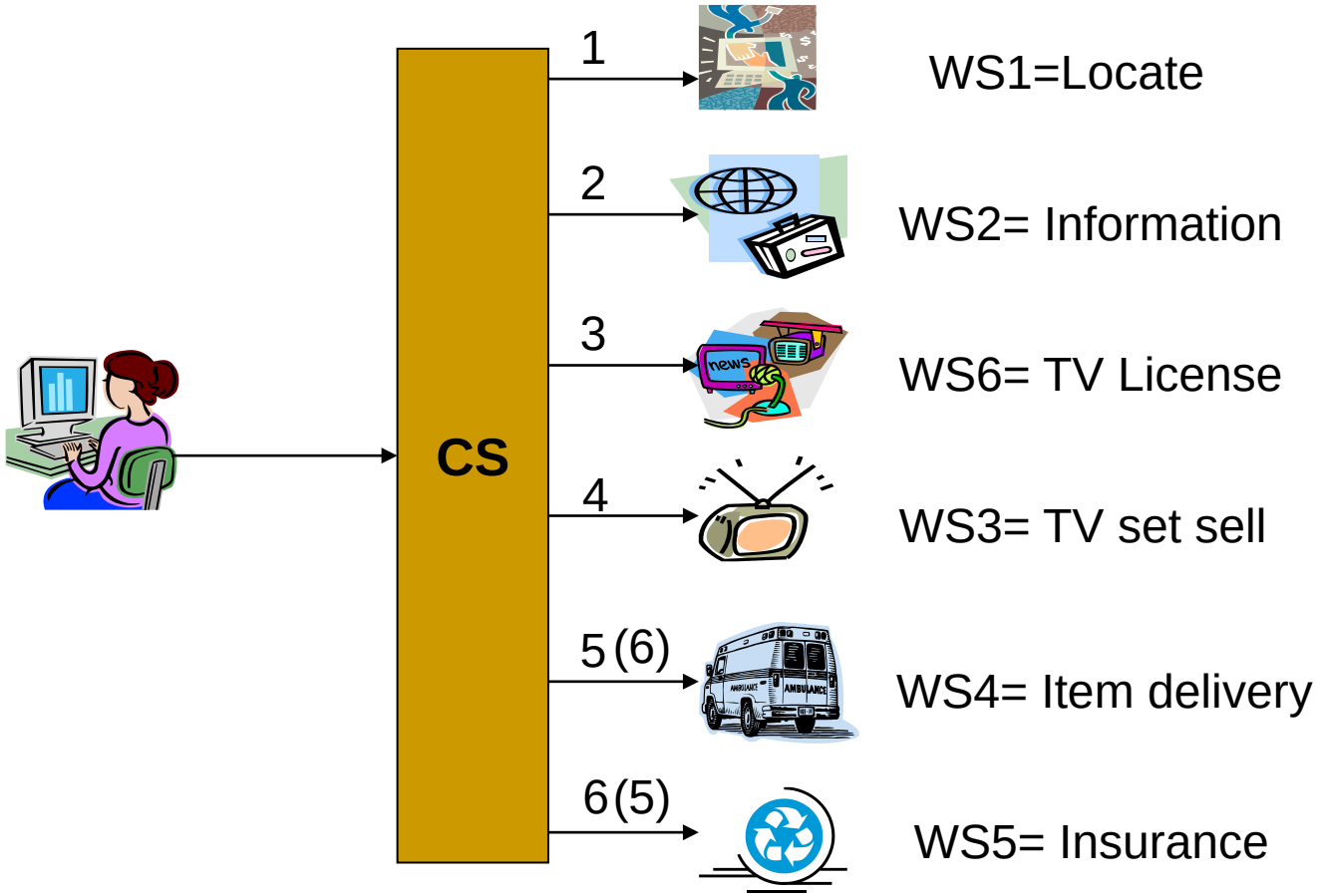
WS3= TV set sell (T) 

WS4= Item delivery (D) 

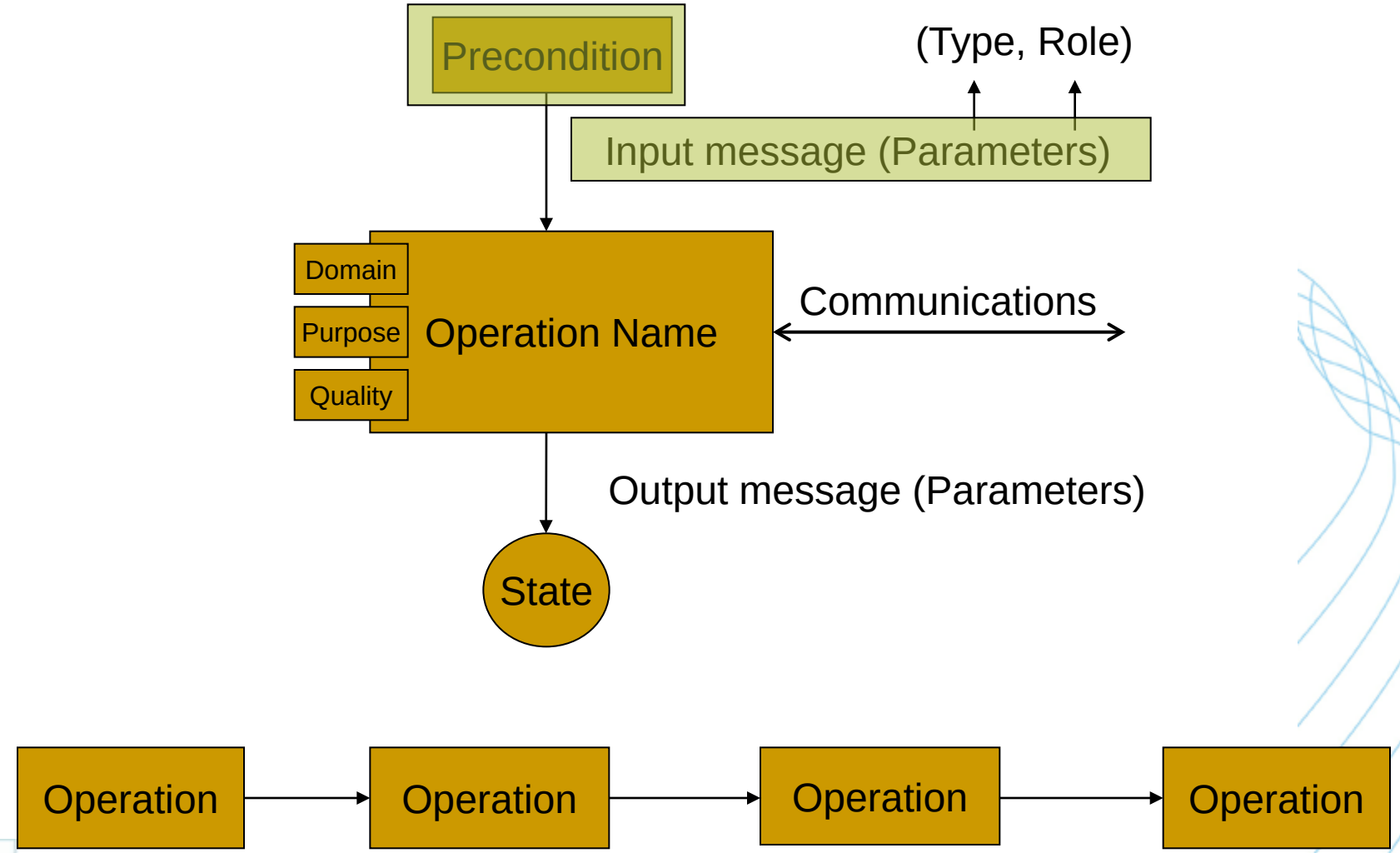
WS5= Insurance (I) 

WS6= TV License 

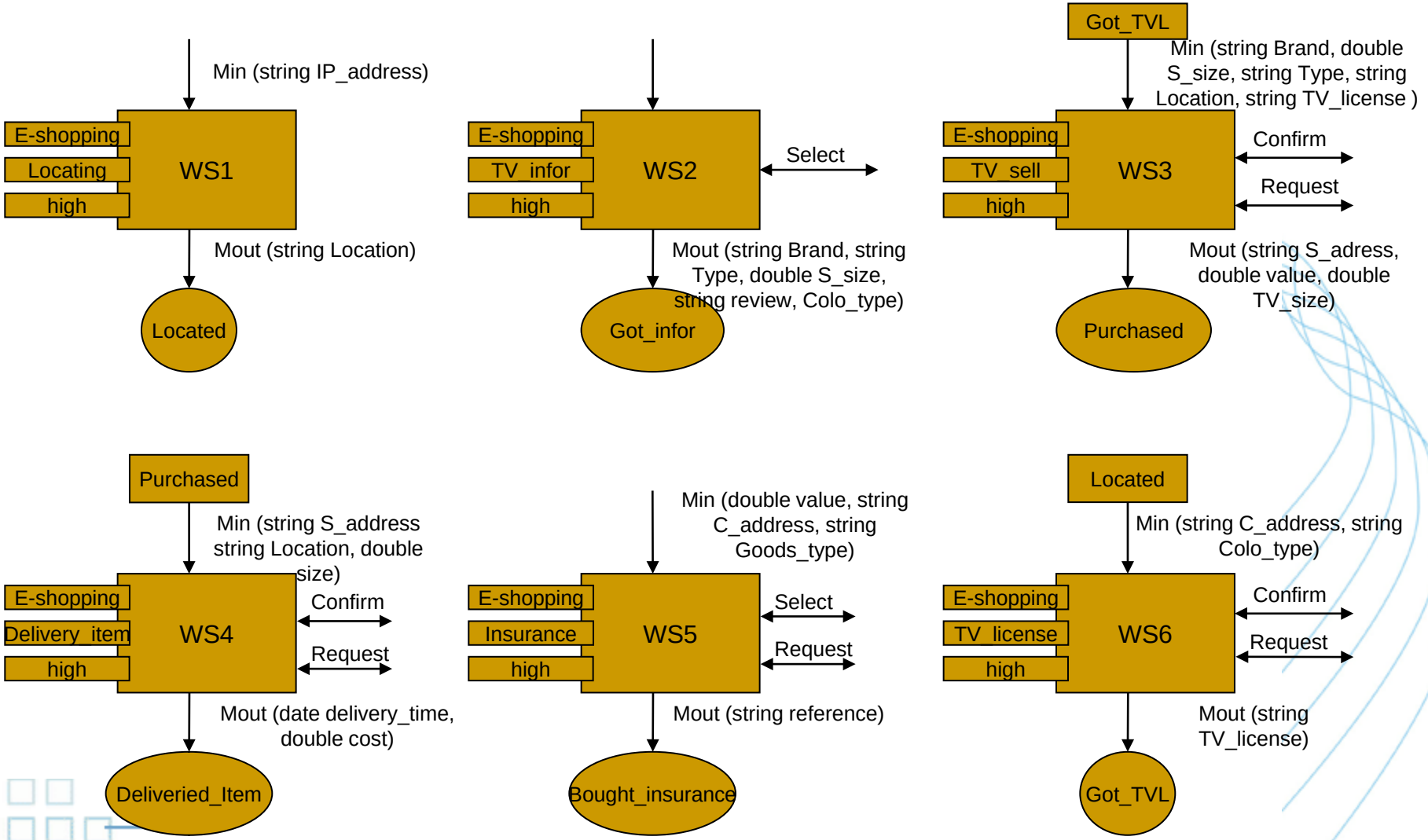
实例


CS
1
2
3
4
5(6)
6(5)
WS1=Locate
WS2= Information
WS6= TV License
WS3= TV set sell
WS4= Item delivery
WS5= Insurance
T ∧◊(D∧I)

Web服务模型



Web服务模型



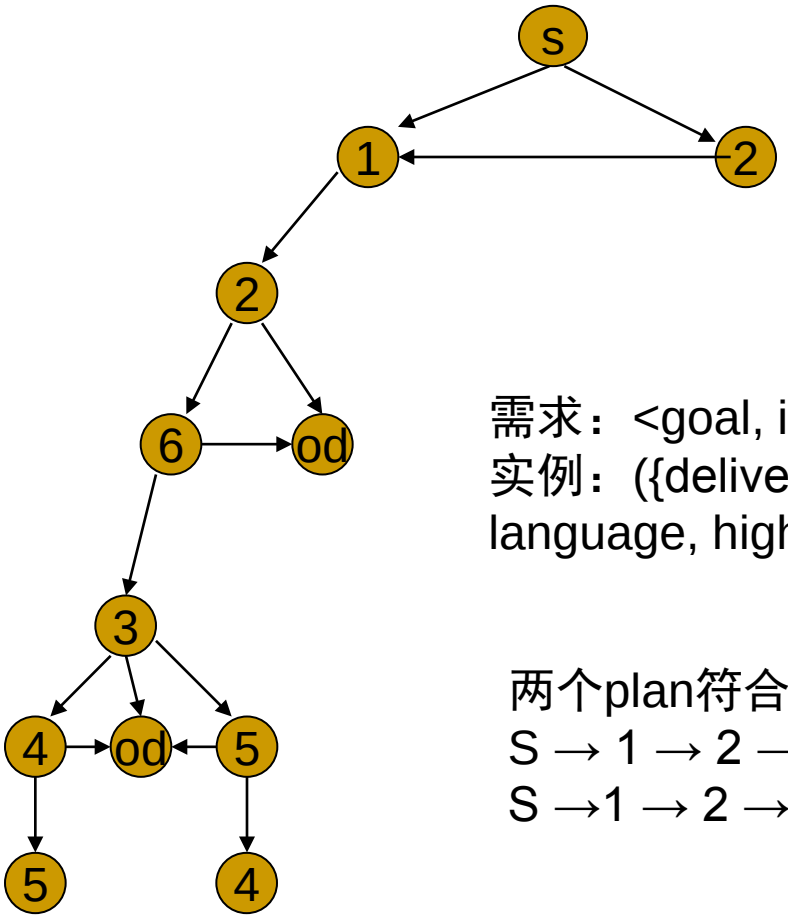
组合问题模型

- 目标规约

$$T \wedge \Diamond(D \wedge I)$$

- 起始条件和数据
  - We are planning from initial operation state
  - The initial knowledge is the information which submitted by Client user
  - Initial state is start
  - Initial knowledge is Customer address, Goods type (TV), IP address

结果



需求：<goal, initial state, domain info, quality info>
实例：({delivered, insured}, {start}, e−shopping, {English language, highsecurity}).

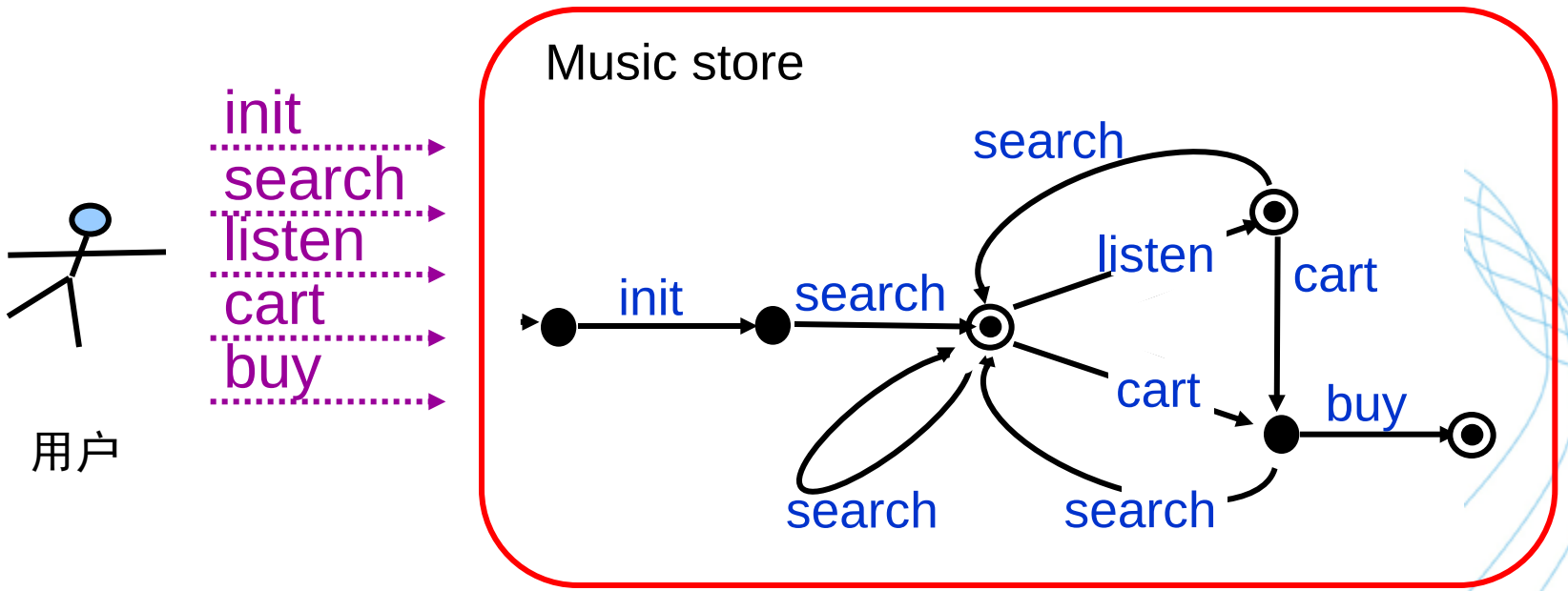
两个plan符合:
S → 1 → 2 → 6 → 3 → 4 → 5
S →1 → 2 → 6 → 3 → 5 → 4

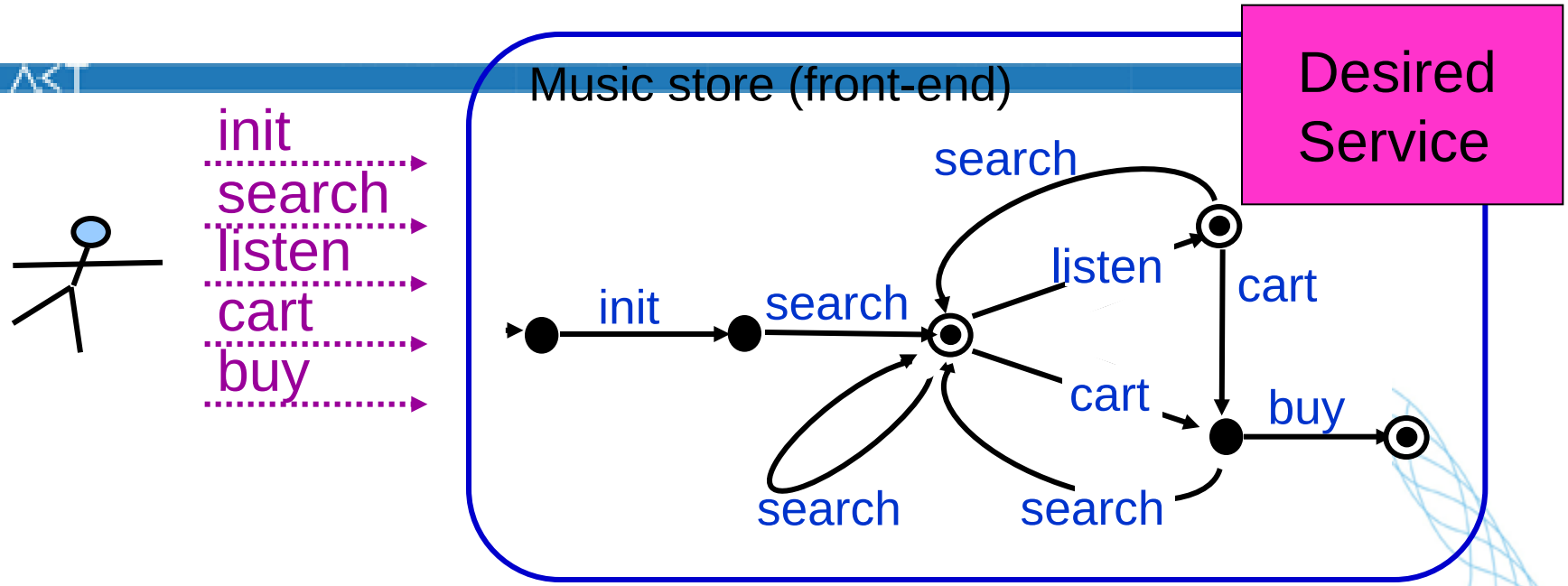
基于自动机的自动服务组合

- 基于自动机模型的服务组合
  - 确定有限自动机
  - guarded 有限自动机
  - 交替自动机

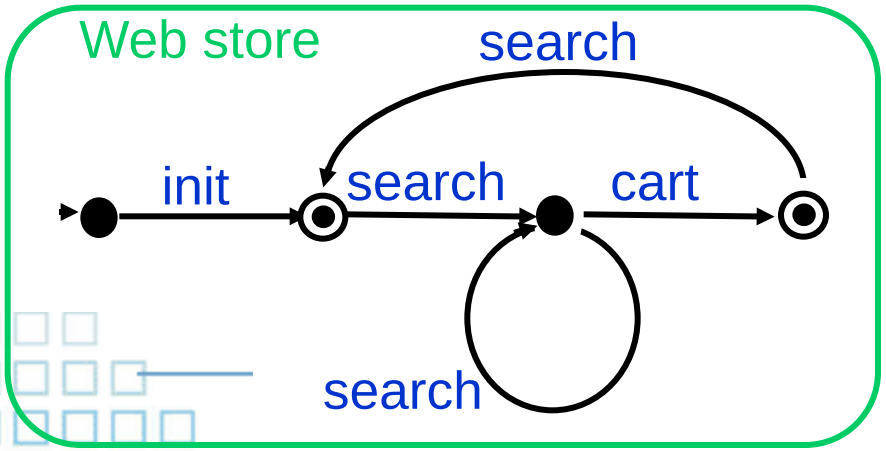
（1）确定型有限自动机模型

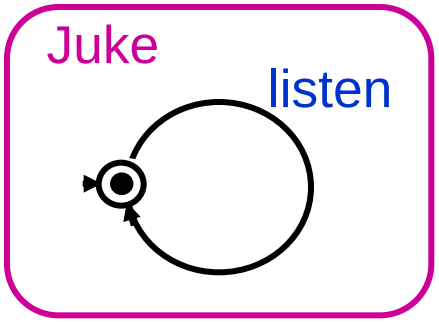
音乐网站

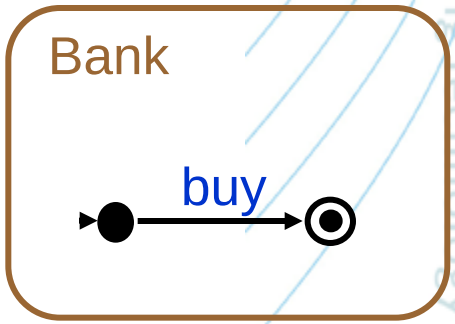




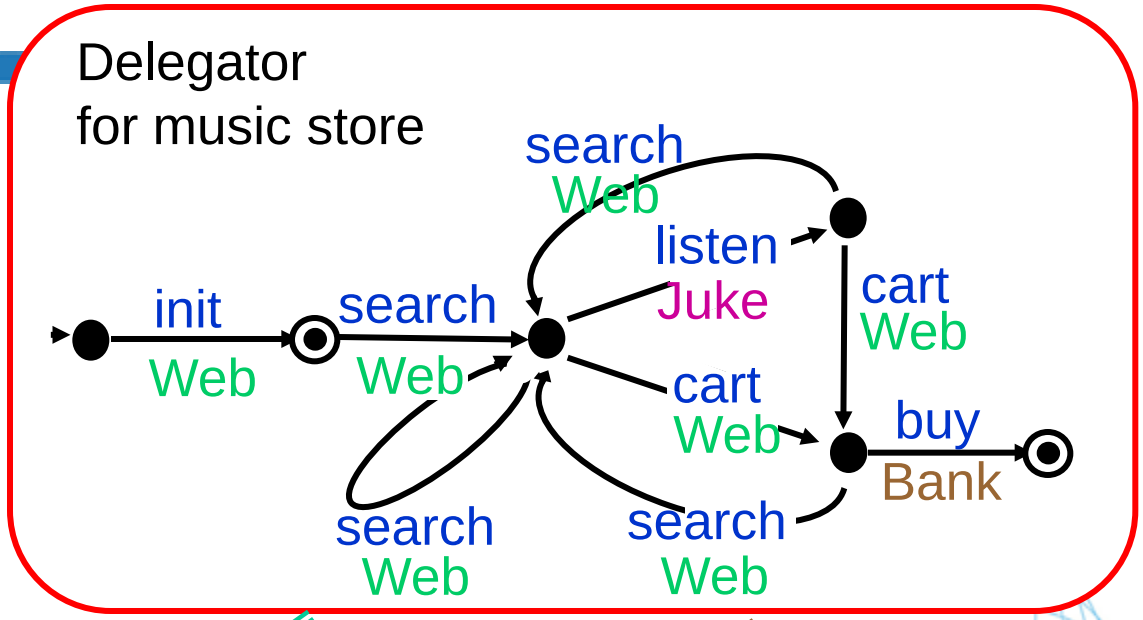
"UDDI++": Available services

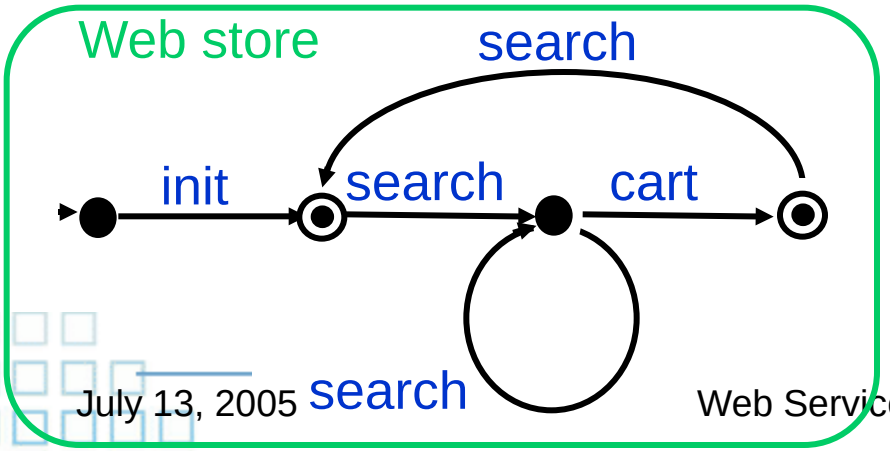


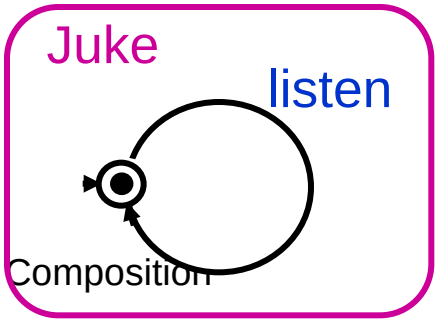


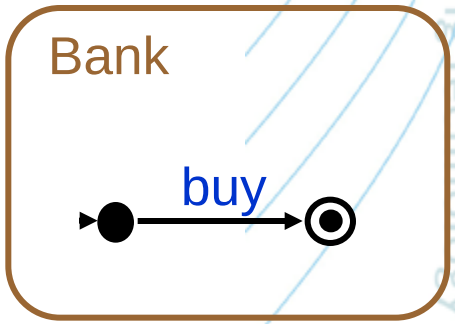


A Roman Composition



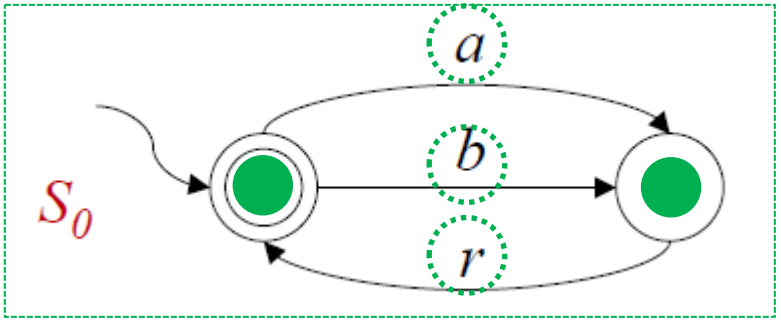




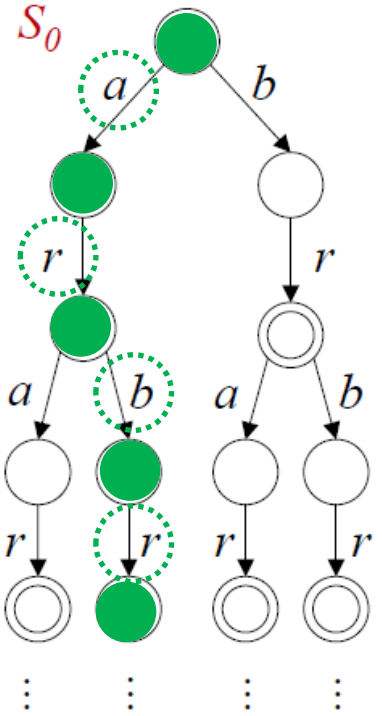


Roman-模型

- 服务：有限状态机（finite state machine)



强调动作序列



每次运行是一个序列

在线音乐服务

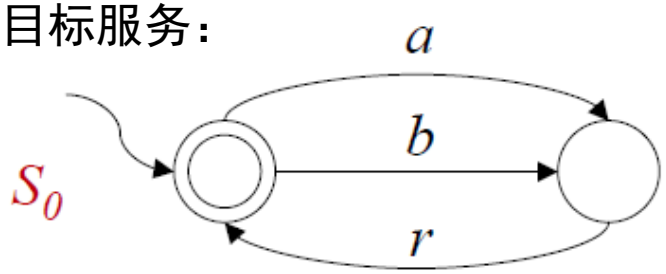
a: "search by author (and select)"
b: "search by title (and select)"
r: "listen (the selected song)"

Roman-模型

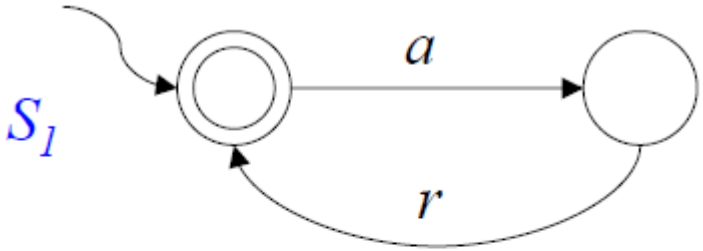
转化为确定型命题动态逻辑公式的可满足性问题

- 服务：有限状态机（finite state machine)

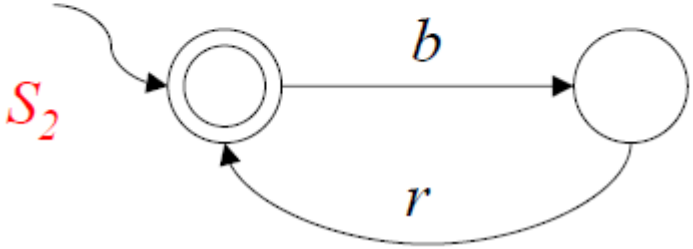
目标服务：



服务1：



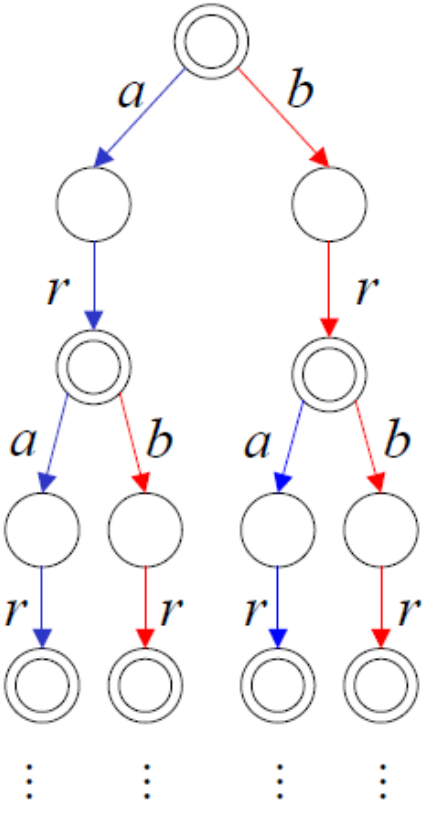
服务2：

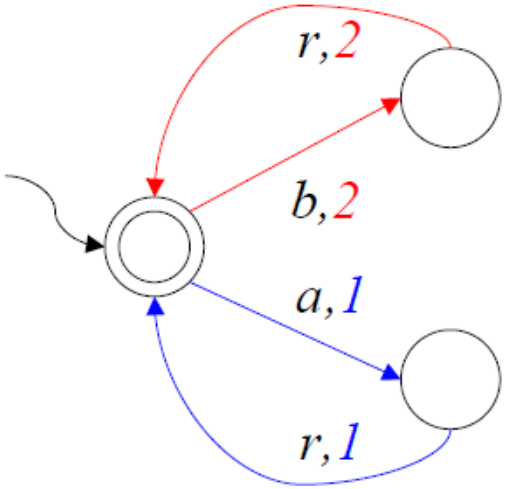


目标服务S_0能否由服务1与服务2组合？

(1) S_0的执行树上的每个动作都可以分配给S_1或S_2的一个动作完成；

(2) S_0的执行树上的动作序列对应到S_1或S_2的动作执行序列的交错连接





问题

- 给定一个目标服务G和一个服务集合S
  - 是否存在S中的服务能组合成G？
  - 如果存在一个组合，是否可以由有限自动机表示？
  - 如果存在，如何找到这个组合？

- 把组合问题转化为命题动态逻辑（DPPL）的可满足性问题
  - DPDL的可满足性问题（EXPTIME-complete）
  - 可以：DPDL的小模型性质
  - 计算DPDL公式的模型

确定型命题动态逻辑DPDL建模

$$\Phi = Init \wedge ([u]\Phi_0 \wedge_{i=1,\ldots,n} [u]\Phi_i \wedge [u]\Phi_{aux})$$

所有服务的初始状态

目标服务

n个组件服务

领域无关的条件

这些DPPL公式的大小关于目标服务或组件服务的大小是多项式的

确定型命题动态逻辑DPPL建模

- 目标服务 $S_0 = (\Sigma, S_0, s^0_0, \delta_0, F_0)$

$s \rightarrow \neg s'$ 状态是两两不相同的

$s \rightarrow \langle a \rangle T \wedge [a]s'$ 刻画状态转移 $s'=\delta_0(s,a)$

$s \rightarrow [a] \perp$ 在状态s上不能执行动作a

$F_0 \equiv \bigvee_{s \in F0} s$ 终止状态

确定型命题动态逻辑DPPL建模

- **组件服务** $S_i = (\Sigma, S_i, s^0_i, \delta_i, F_i)$

$s \rightarrow \neg s'$ 状态是两两不相同的

如果Si执行a则到新状态s', 否则状态保持不变

$s \rightarrow [a](moved_i \wedge s' \vee \neg moved_i \wedge s)$ 刻画状态转移$s'=\delta_i(s,a)$

$s \rightarrow [a](\neg moved_i \wedge s)$ 当动作a要被执行，而服务S_i不能执行a

$F_i \equiv \bigvee_{s \in Fi} s$ 终止状态

确定型命题动态逻辑DPPL建模

- 其他条件 Φ_{aux}

$<a>T \rightarrow [a] \vee_{i=1,\ldots,n} moved_i$ 对于动作a，至少有一个服务执行a

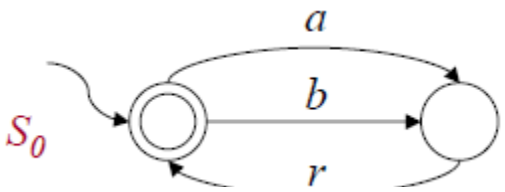
$F_0 \rightarrow \wedge_{i=1,\ldots,n} F_i$ 当目标服务到达终止状态时，所有组件服务都到达终止状态

  - 初始状态

$Init \equiv s^0_0 \wedge_{i=1\ldots n} s^0_i$ 目标服务和所有组件服务都处于初始状态

例子

- 目标服务



$s_0^0 \rightarrow \neg\, s_0^1$

$s_0^0 \rightarrow \langle a \rangle T \wedge [a]\, s_0^1$

$s_0^0 \rightarrow \langle b \rangle T \wedge [b]\, s_0^1$

$s_0^1 \rightarrow \langle r \rangle T \wedge [r]\, s_0^0$

$s_0^0 \rightarrow [r] \perp \wedge [r]\, s_0^0$

$s_0^1 \rightarrow [a] \perp$

$s_0^1 \rightarrow [b] \perp$

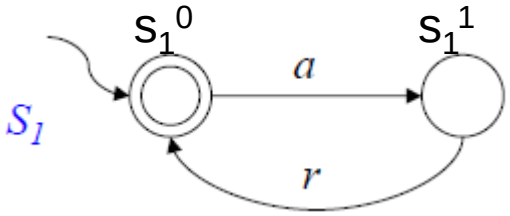
$F_0 \equiv s_0^0$

...

...

...

- 组件服务



$s_1^0 \rightarrow \neg\, s_1^1$

$s_1^0 \rightarrow [a]\,(moved_1 \wedge s_1^1 \vee \neg\, moved_1 \wedge s_1^0)$

$s_1^0 \rightarrow [r] \neg\, moved_1 \wedge s_1^0$

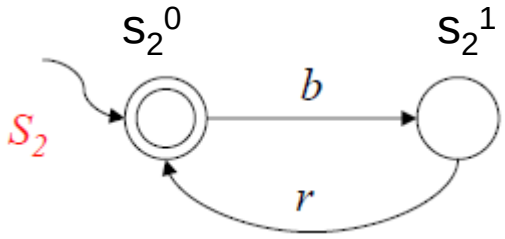
$s_1^0 \rightarrow [b] \neg\, moved_1 \wedge s_1^0$

$s_1^1 \rightarrow [a] \neg\, moved_1 \wedge s_1^1$

$s_1^1 \rightarrow [b] \neg\, moved_1 \wedge s_1^1$

$s_1^1 \rightarrow [r]\,(moved_1 \wedge s_1^0 \vee \neg\, moved_1 \wedge s_1^0)$

$F_1 \equiv s_1^0$



$s_2^0 \rightarrow \neg\, s_2^1$

$s_2^0 \rightarrow [b]\,(moved_2 \wedge s_2^1 \vee \neg\, moved_2 \wedge s_2^0)$

$s_2^0 \rightarrow [r] \neg\, moved_2 \wedge s_2^0$

$s_2^0 \rightarrow [a] \neg\, moved_2 \wedge s_2^0$

$s_2^1 \rightarrow [b] \neg\, moved_2 \wedge s_2^1$

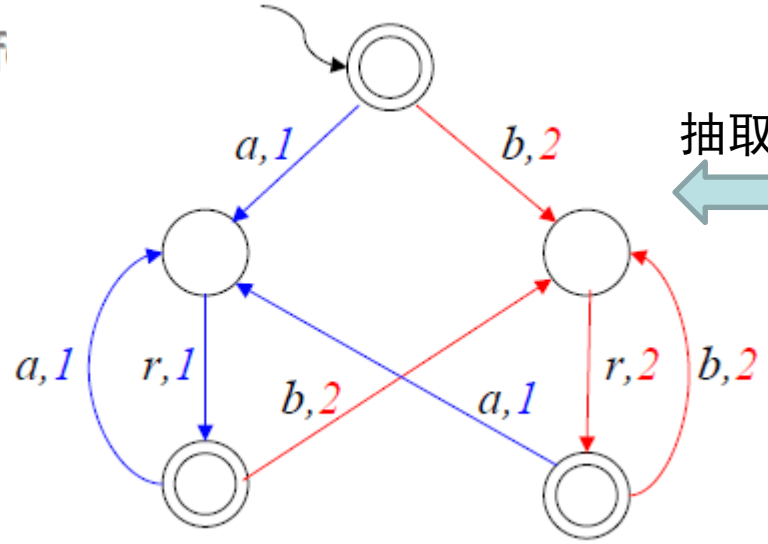
$s_2^1 \rightarrow [a] \neg\, moved_2 \wedge s_2^1$

$s_2^1 \rightarrow [r]\,(moved_2 \wedge s_2^0 \vee \neg\, moved_2 \wedge s_2^0)$

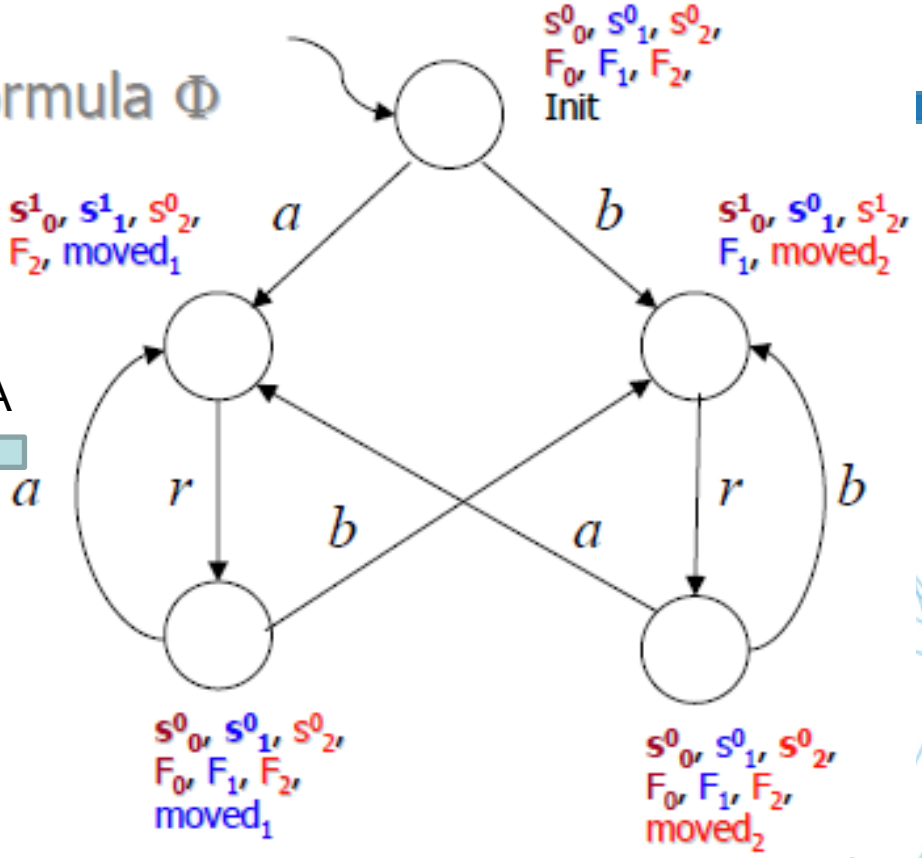
$F_2 \equiv s_2^0$

Check: run SAT on DPDL formula Φ

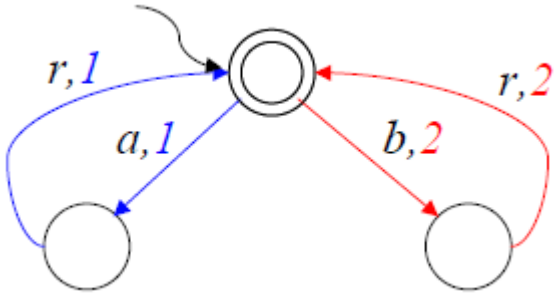
Yes ⇒ (small) model



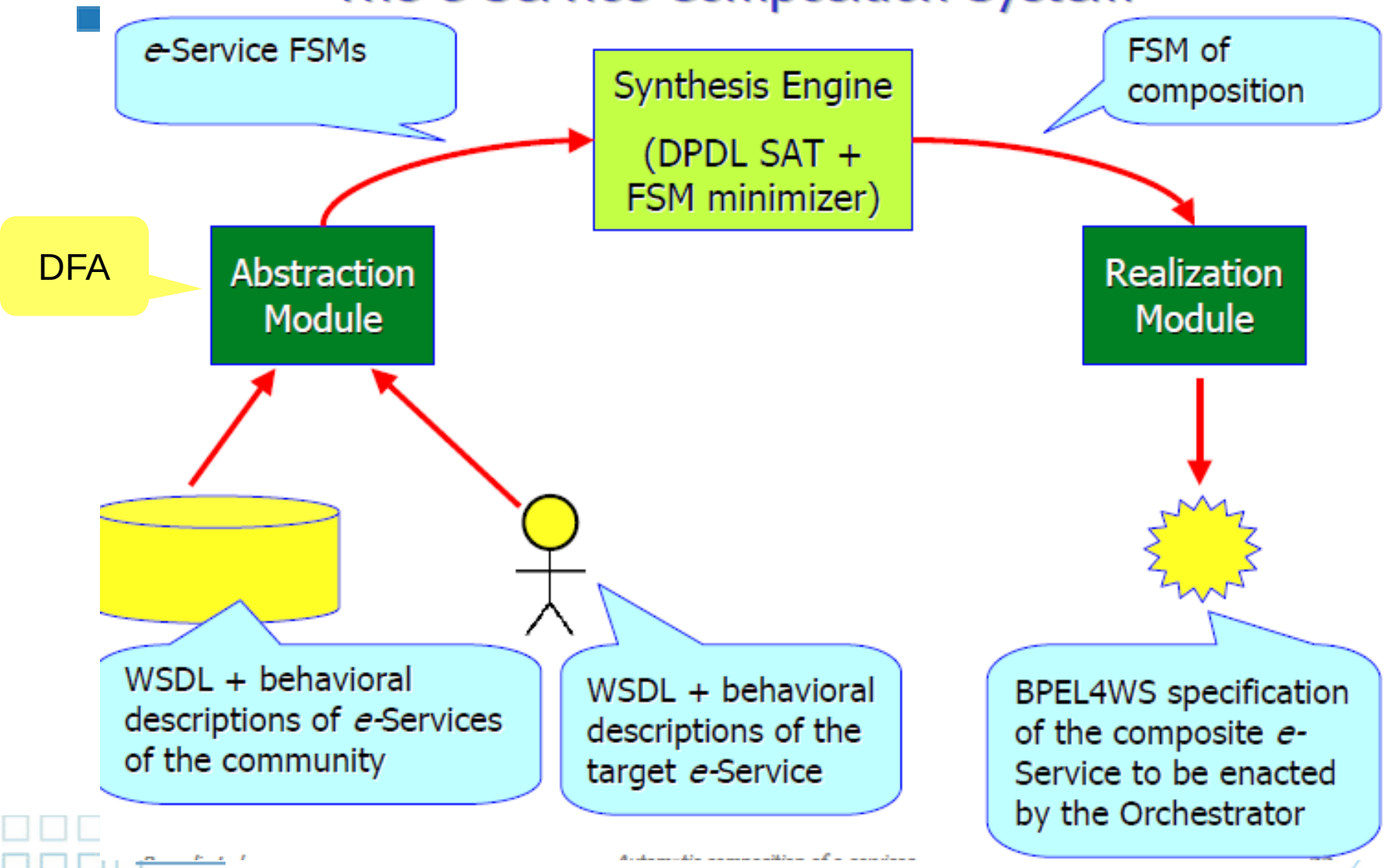
抽取FSA



极小化FSA



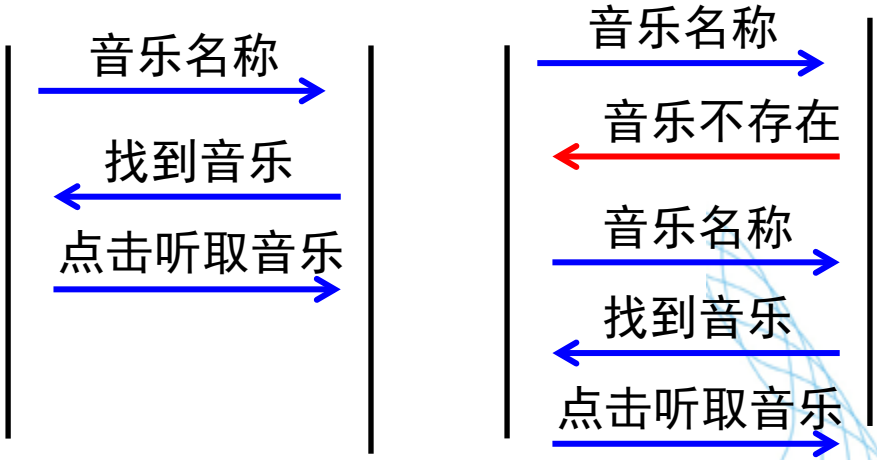
The *e*-Service Composition System

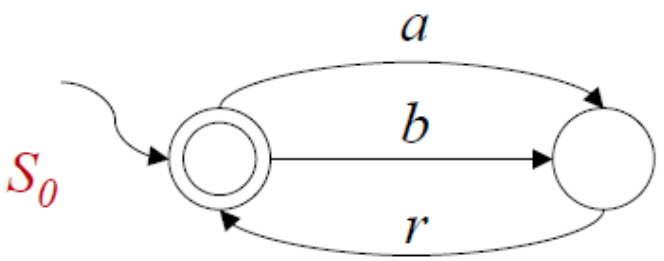


（2）Guarded 自动机

确定有限自动机

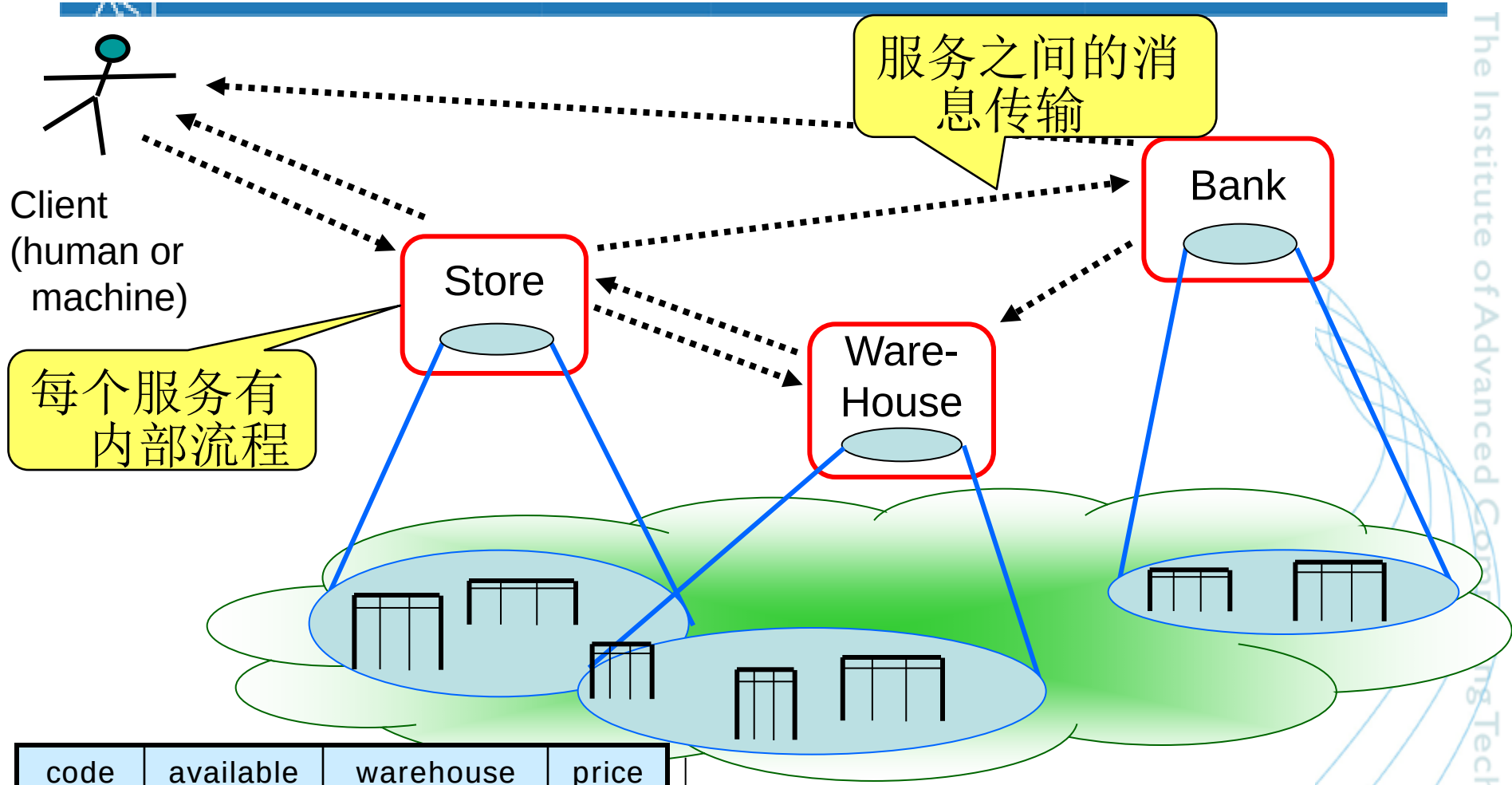
- 把服务的一个功能建模为自动机的一个动作
- 未考虑
  - 服务之间的消息交互
  - 服务功能实现的条件





a: "search by author (and select)"
b: "search by title (and select)"
r: "listen (the selected song)"

服务组合场景

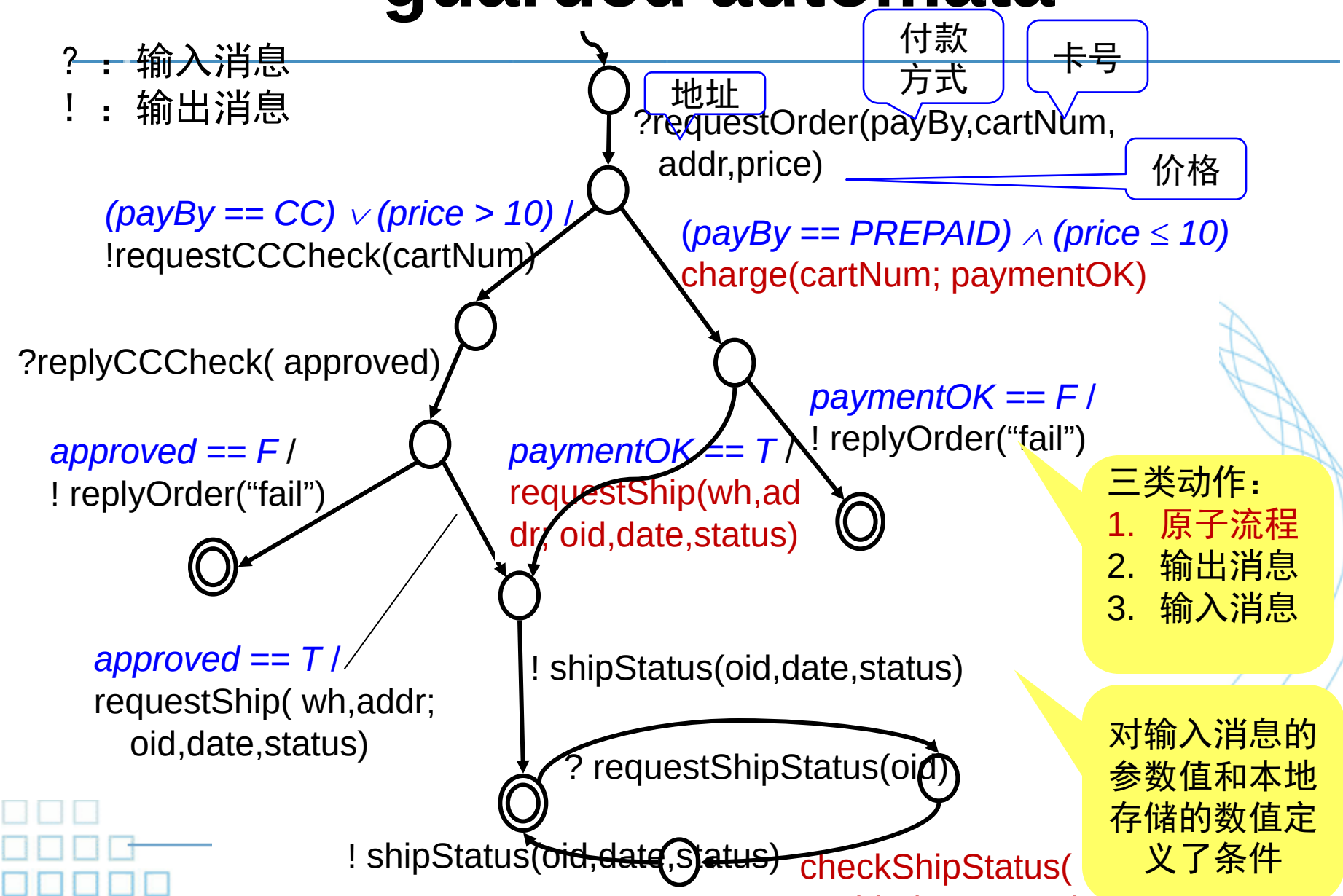


code	available	warehouse	price
HP15	T	NGW	5
HS72	F	SW	10

每个服务都有本地数据库

guarded automata

? ： 输入消息
! ： 输出消息



三类动作：
1. 原子流程
2. 输出消息
3. 输入消息

对输入消息的参数值和本地存储的数值定义了条件

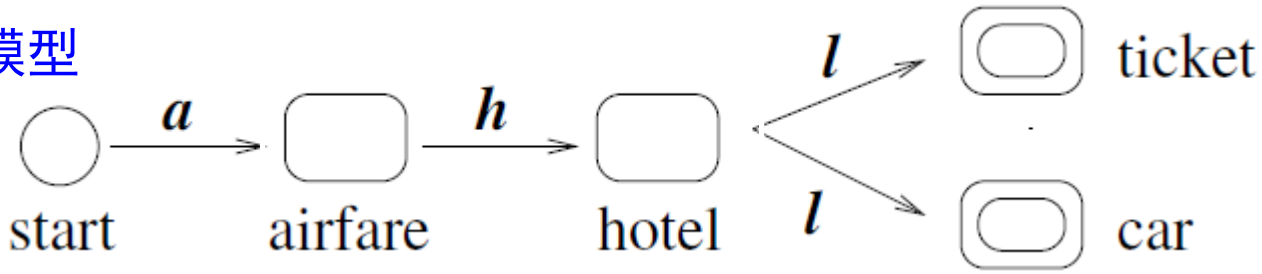
（3）交替自动机
(Finite Alternating automata)

例： 旅游代理

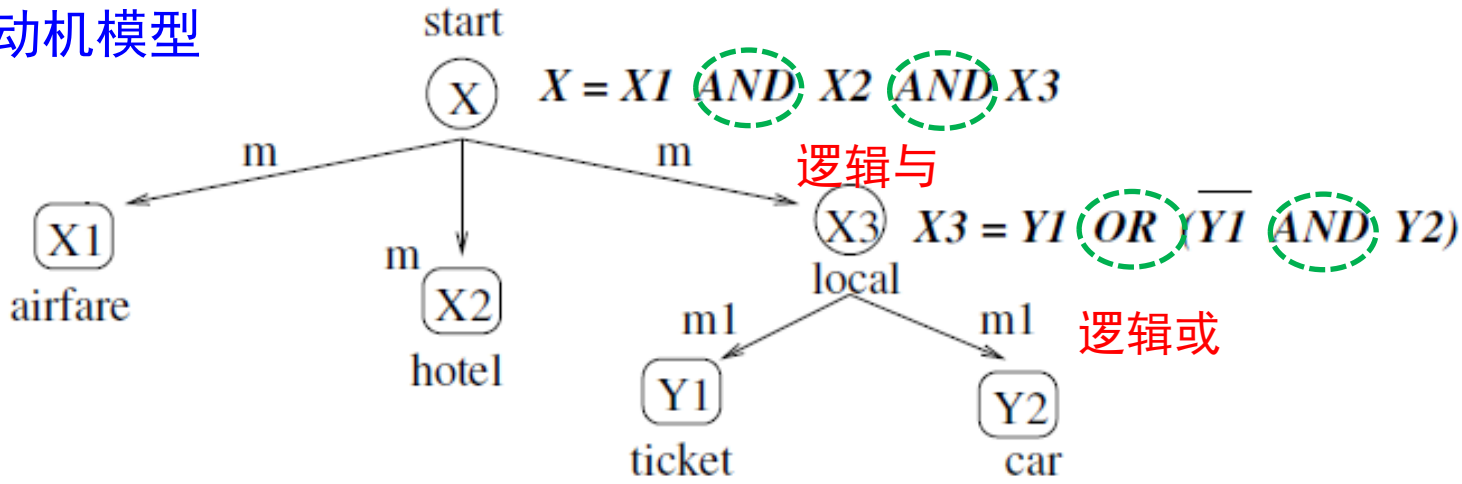
- 一个提供到奥兰多迪斯尼乐园游玩的旅游预定
  - 机票
  - 旅馆
  - 迪斯尼门票或有折扣的出租车预订

确定/非确定自动无法刻画并发

自动机模型



交替自动机模型



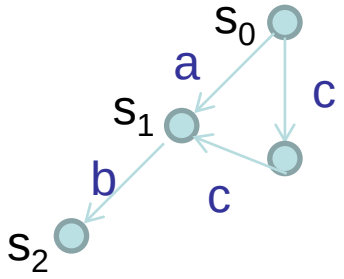
交替自动机

- $A=(\Sigma, S, s_0, \delta, F)$
  - **Σ:** 非空有限输入字符表
  - **S:** 非空有限状态集合
  - **δ:** $S \times \Sigma \rightarrow B^+(S)$，其中，$B^+(S)$是S上一个布尔公式的集合
  - **F:** 非空终止状态集合

$\delta(s, a)=(s_1 \wedge s_2) \vee (s_3 \wedge s_4)$

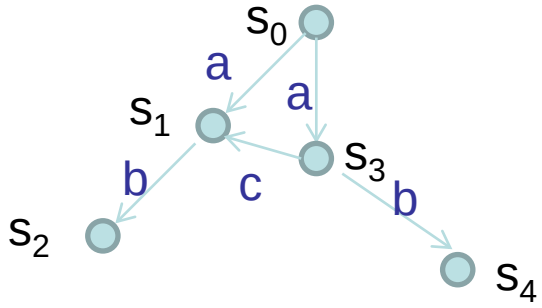
非确定自动机：$\delta(s, a)=\{s_1, s_2\} = s_1 \vee s_2$

确定自动机A



$ab \in L(A)$ iff s_2是终止状态

非确定自动机A



$ab \in L(A)$ iff s_2或s_4是终止状态

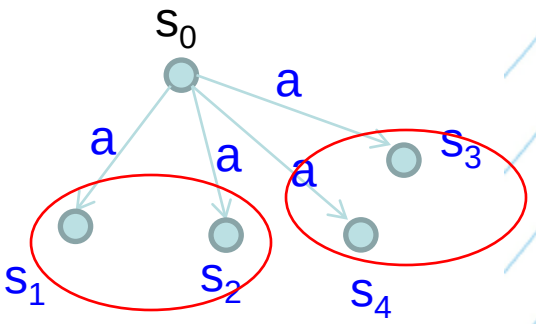
交替自动机

- $A=(\Sigma, S, s_0, \delta, F)$
  - Σ: 非空有限输入字符表
  - S：非空有限状态集合
  - S_0: 初始状态
  - δ：$S \times \Sigma \rightarrow B^+(S)$，其中，$B^+(S)$是S上一个布尔公式的集合
  - F: 非空终止状态集合

- 交替自动机A在字符串$a_1a_2\cdots a_n$上的运行是一棵树

$\delta(s_0, a)=(s_1 \wedge s_2) \vee (s_3 \wedge s_4)$

A接受a当且仅当s_1与s_2都是终止状态，或s_3与s_4都是终止状态



定义$f(s_i)=1$ 如果 s_i是终止状态；否则 $f(s_i)=0$

⇒

自底向上判断：

A接受a当且仅当

$(f(s_1) \wedge f(s_2)) \vee (f(s_3) \wedge f(s_4))=1$

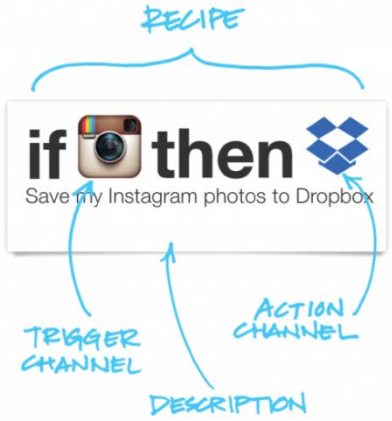
服务组合的类型

- Web服务组合
  - 手动：静态组合
  - 半自动：动态组合
  - 全自动组合

- RESTful服务/Web API组合

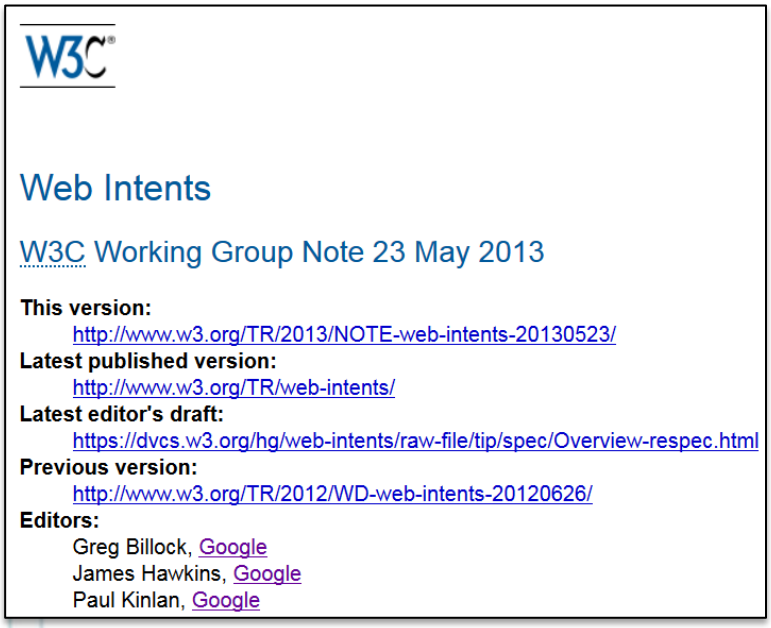
IFTTT

- 任何一个服务（App或者网站应用）都有可访问的URI
- IFTTT(If This Then That)可以为不同服务间设置条件语句链
  - 本身是集中式服务和App



Web Intents

- 浏览器模块，支持浏览器中不同网页之间的参数传递，实现服务之间的协作组合
- W3C的一个建议标准，Chrome原生支持
- IE 8+， firefox 3+通过js支持



W3C

Web Intents

W3C Working Group Note 23 May 2013

This version:
http://www.w3.org/TR/2013/NOTE-web-intents-20130523/
Latest published version:
http://www.w3.org/TR/web-intents/
Latest editor's draft:
https://dvcs.w3.org/hg/web-intents/raw-file/tip/spec/Overview-respec.html
Previous version:
http://www.w3.org/TR/2012/WD-web-intents-20120626/
Editors:
Greg Billock, Google
James Hawkins, Google
Paul Kinlan, Google

Share a link

http://paul.kinlan.me | Share link

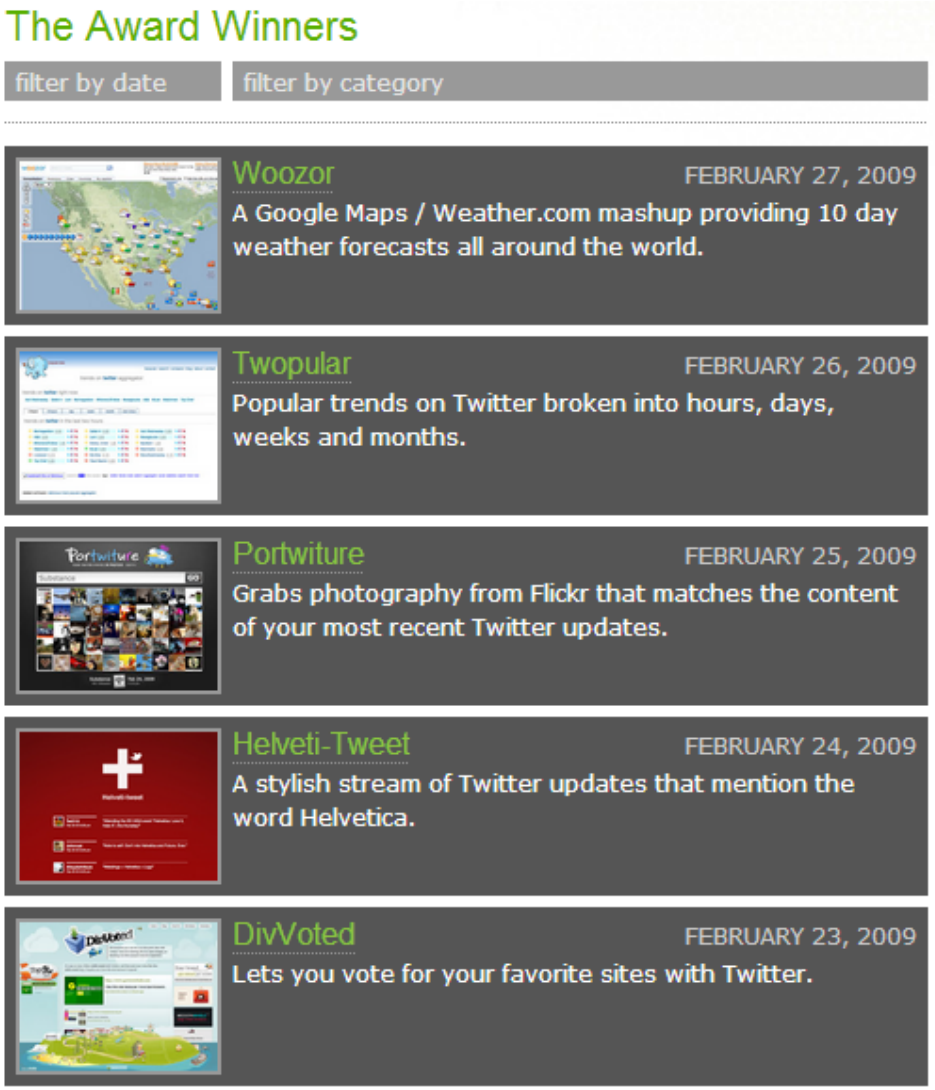
```
var intent = new Intent(
    "http://webintents.org/share",
    "text/uri-list",
    [ url ]);

window.navigator.startActivity(intent);
```

Mashup/Web Mashup

- 中文：混搭、糅合
- **A mashup, in web development, is a web page, or web application, that uses and combines data, presentation or functionality from two or more sources to create new services.-----*From Wikipedia*.**
- **Web2.0的典型应用之一，一种将不同来源的数据和功能无缝组合，形成全新、集成式服务的模式**
- 主要的技术特征
  - **combination, visualization, aggregation**
  - **Easy and fast integration**

Web Mashup的代表性应用

- 地图 mashups
  - Google Maps, Yahoo Maps, Microsoft Virtual Earth
- 视频与图片mashups
  - Flickr, Youtube
- 搜索与购物mashups
  - eBay, Amazon
- 新闻 mashups
  - Diggdot.us= Digg.com + Slashdot.org + Del.icio.us



新闻Mashup实例

- **News+Map**

★★★★☆
Click stars to vote

Description

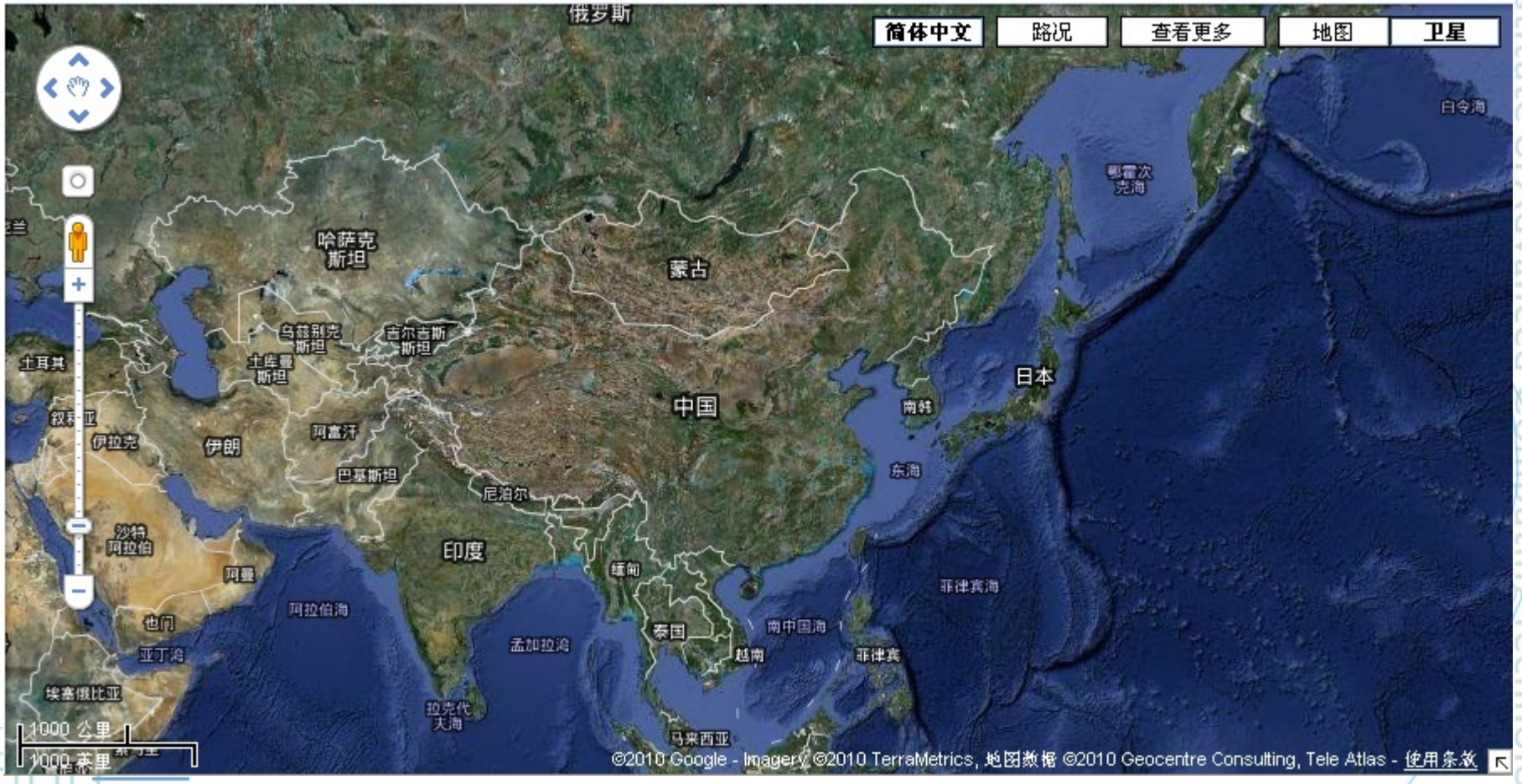
See where the latest news is happening in the UK.

APIs BBC + Google Maps
Tags mapping, news, uk

Added 22 Jan 2006
Who boneill [Profile]
URL http://dev.benedictoneill.com ...

国内外研究现状

- 开发平台发展历史



国内外研究现状

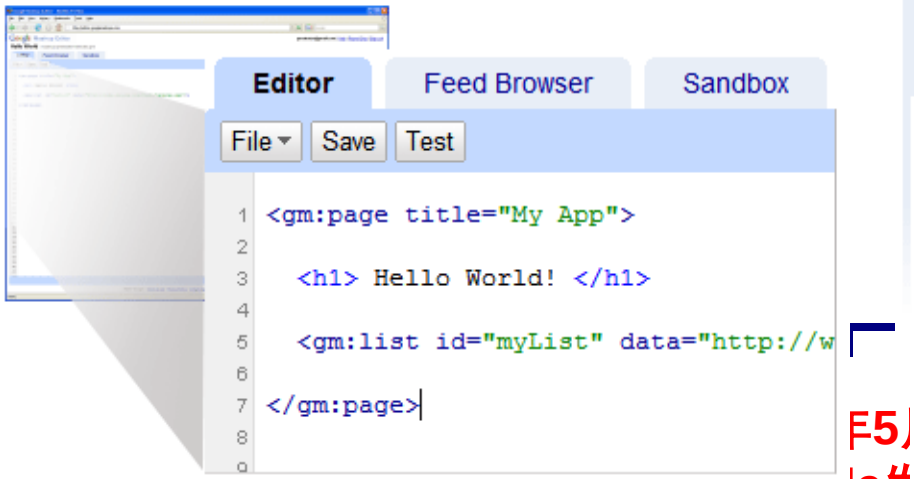
- 开发平台发展历史

国内外研究现状

- 开发平台发展历史

国内外研究现状

- 开发平台发展历史



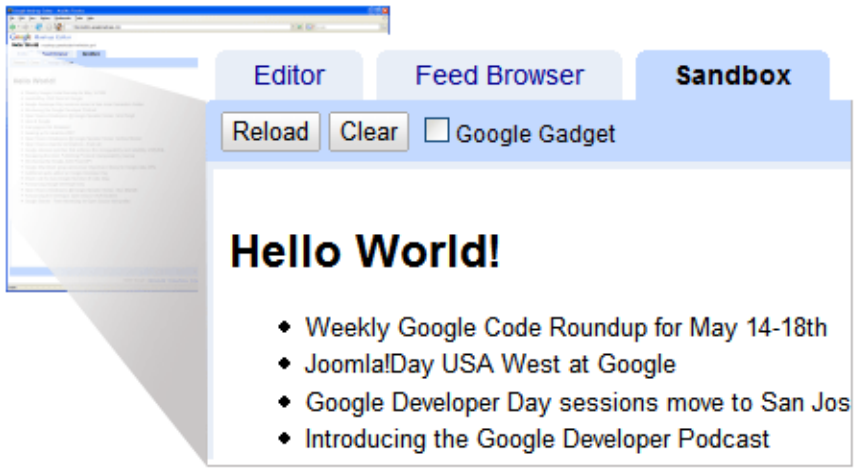
Yahoo! pipes

Google

Editor测

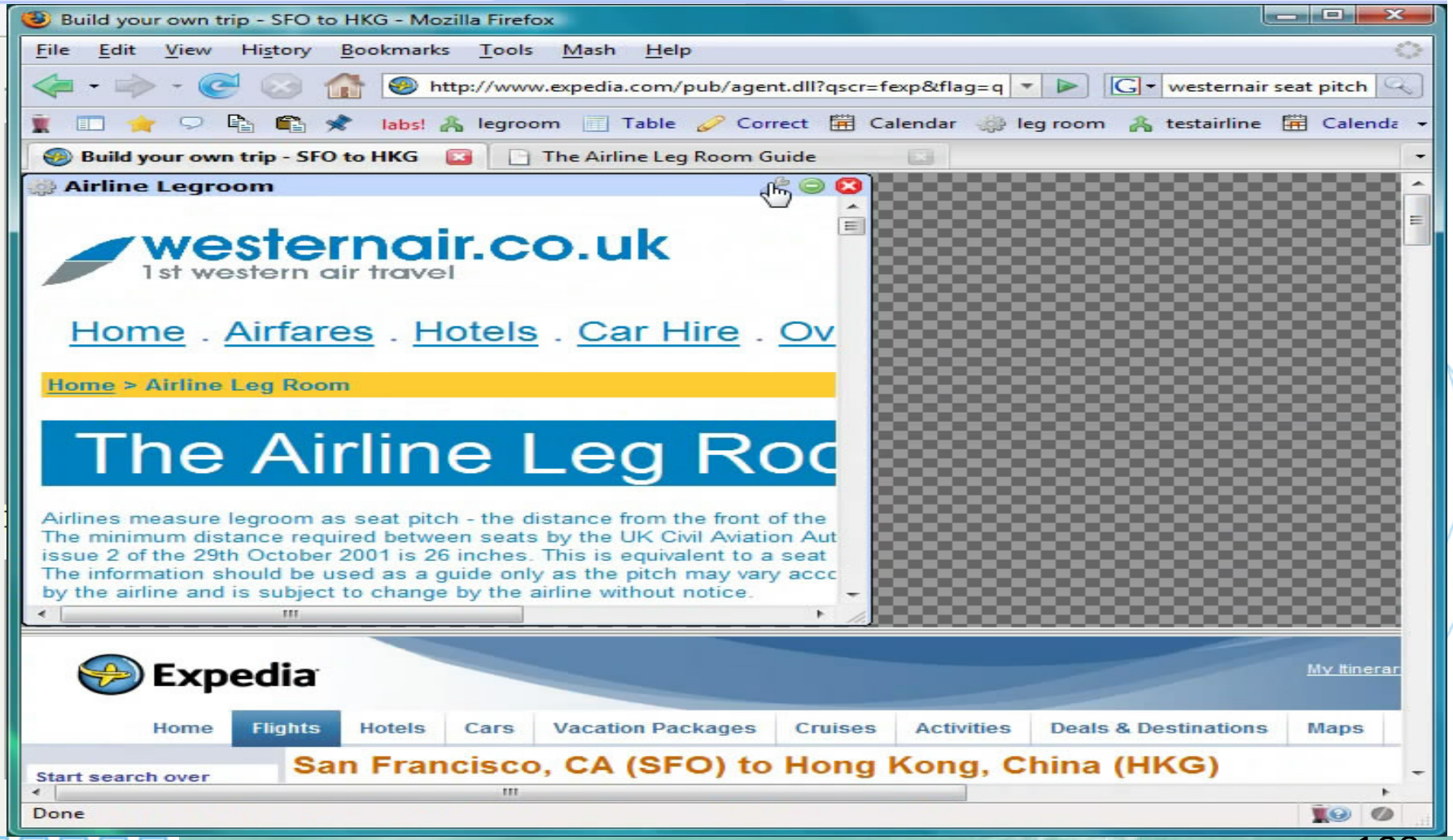
持在线的

创建、调



国内外研究现状

- 开发平台发展历史



国内外研究现状

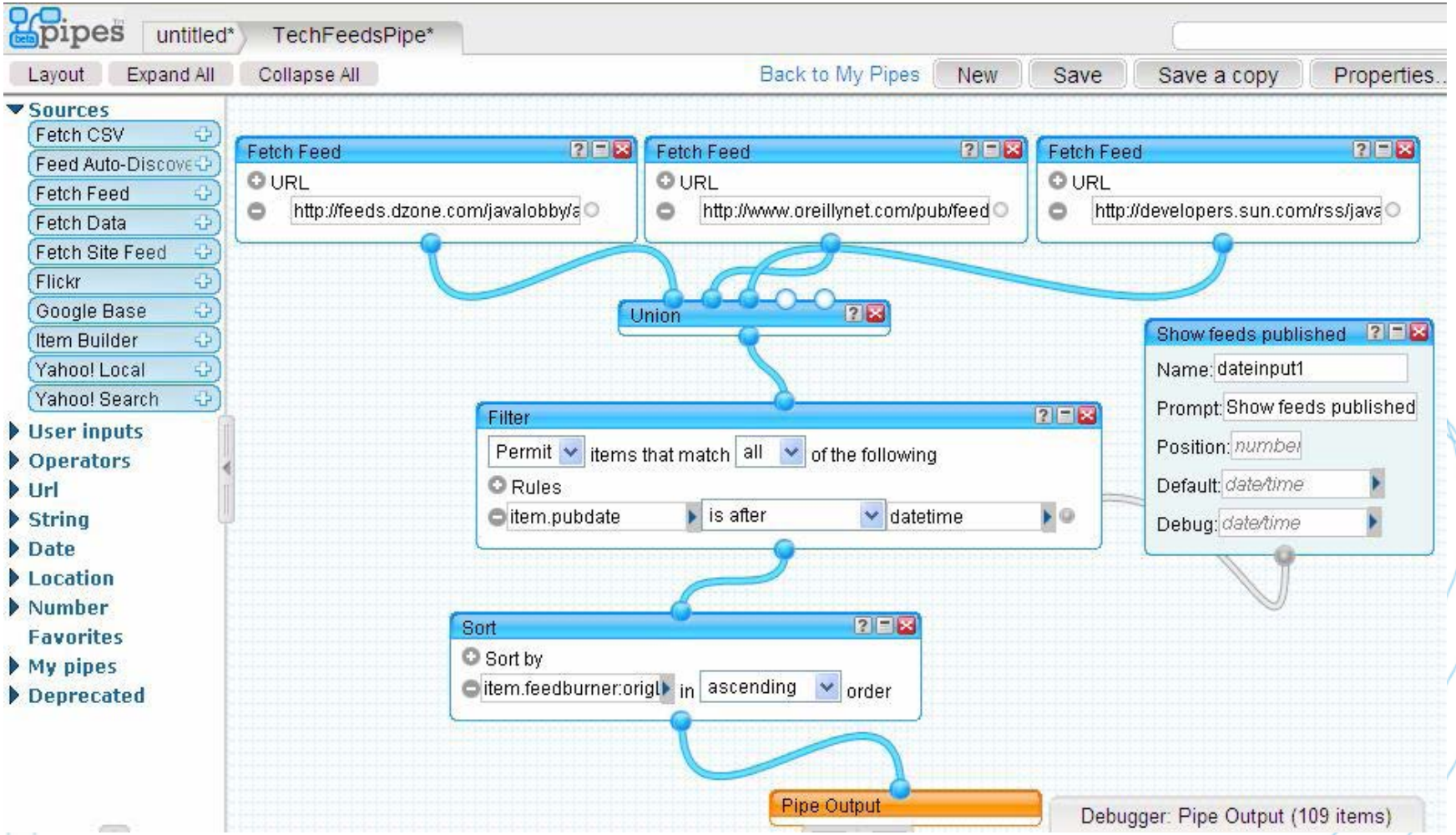
- 开发平台发展历史

国内外研究现状

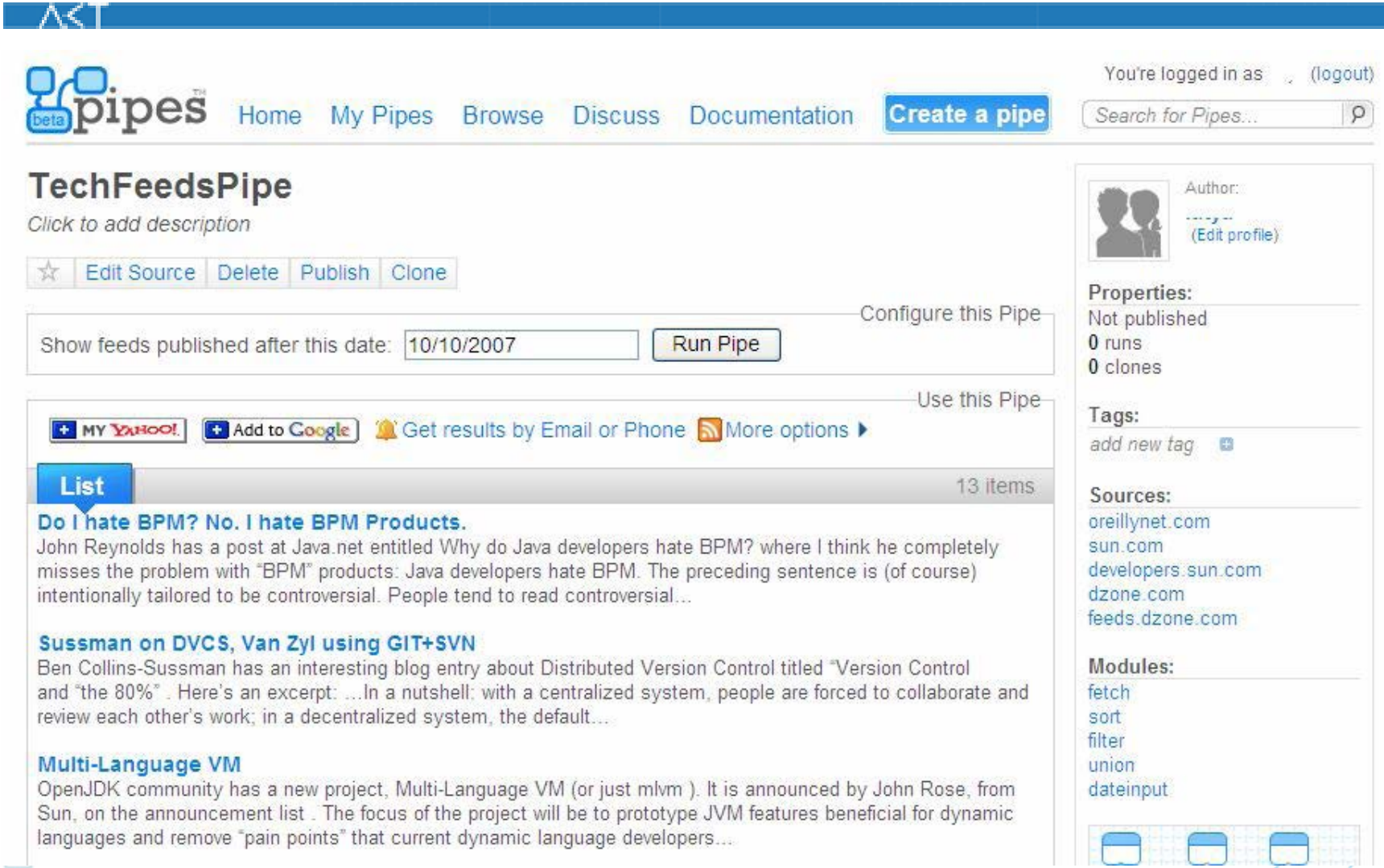
- 各大厂商推出的**Mashup**构建平台比较

	Yahoo！Pipes	GME	Microsoft Popfly	Intel Mash Maker	IBM Mashup Center
开发界面	拖拽方式	文本编辑	拖拽、文本	鼠标操作	拖拽方式
目标人群	高级Web用户	程序员	普通web用户	普通web用户	企业用户
系统需求	普通浏览器	普通浏览器	SilverLight	Mash Maker插件	WebSphere Application Server
内容类型	数据、逻辑	数据、逻辑、界面	数据、逻辑、界面	数据	数据、界面
其他			重数据显示轻数据操作	目标提升用户浏览网页体验	

Yahoo! Pipe : Example



Yahoo! Pipe : Example



国内外研究现状

- **IBM Mashup Center**

图 1. InfoSphere MashupHub 的架构

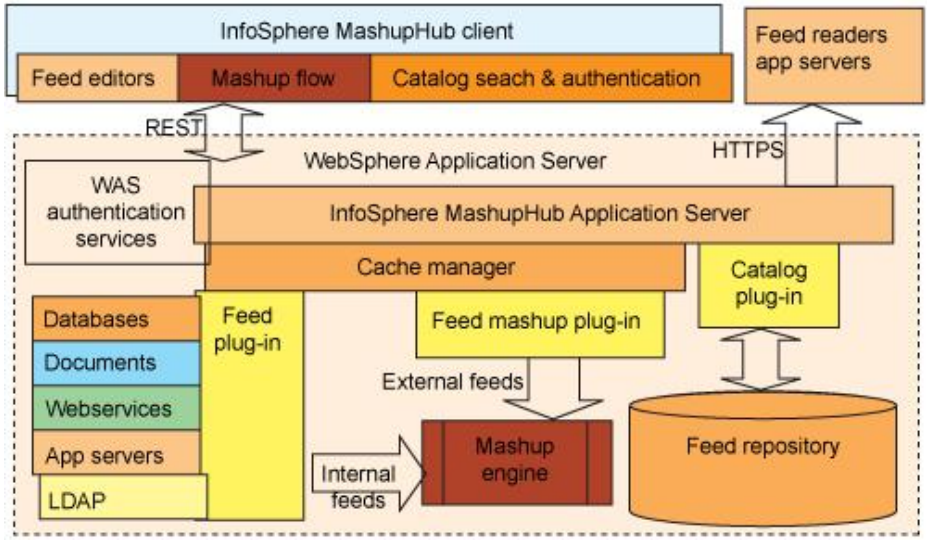


图 2. IBM Mashup Center的组成

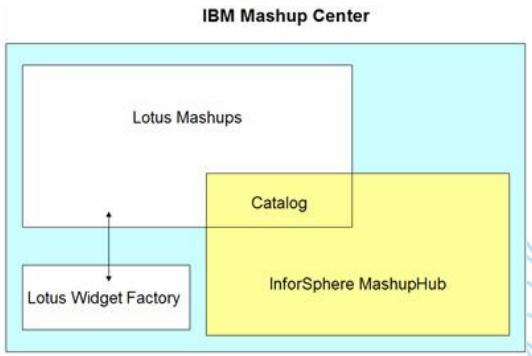
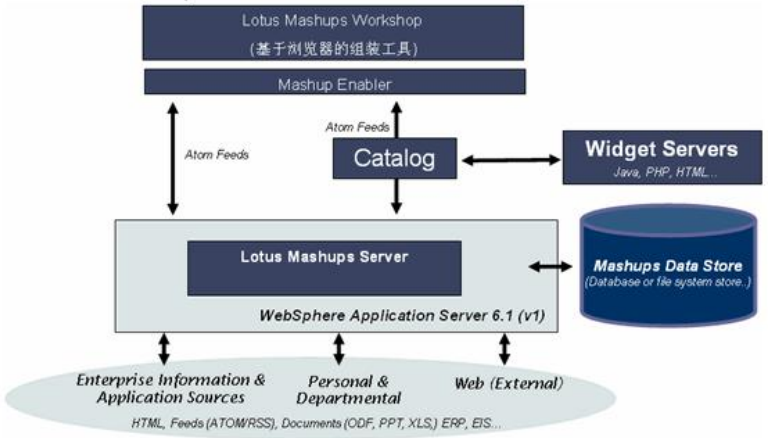


图 1. Lotus Mashups 架构

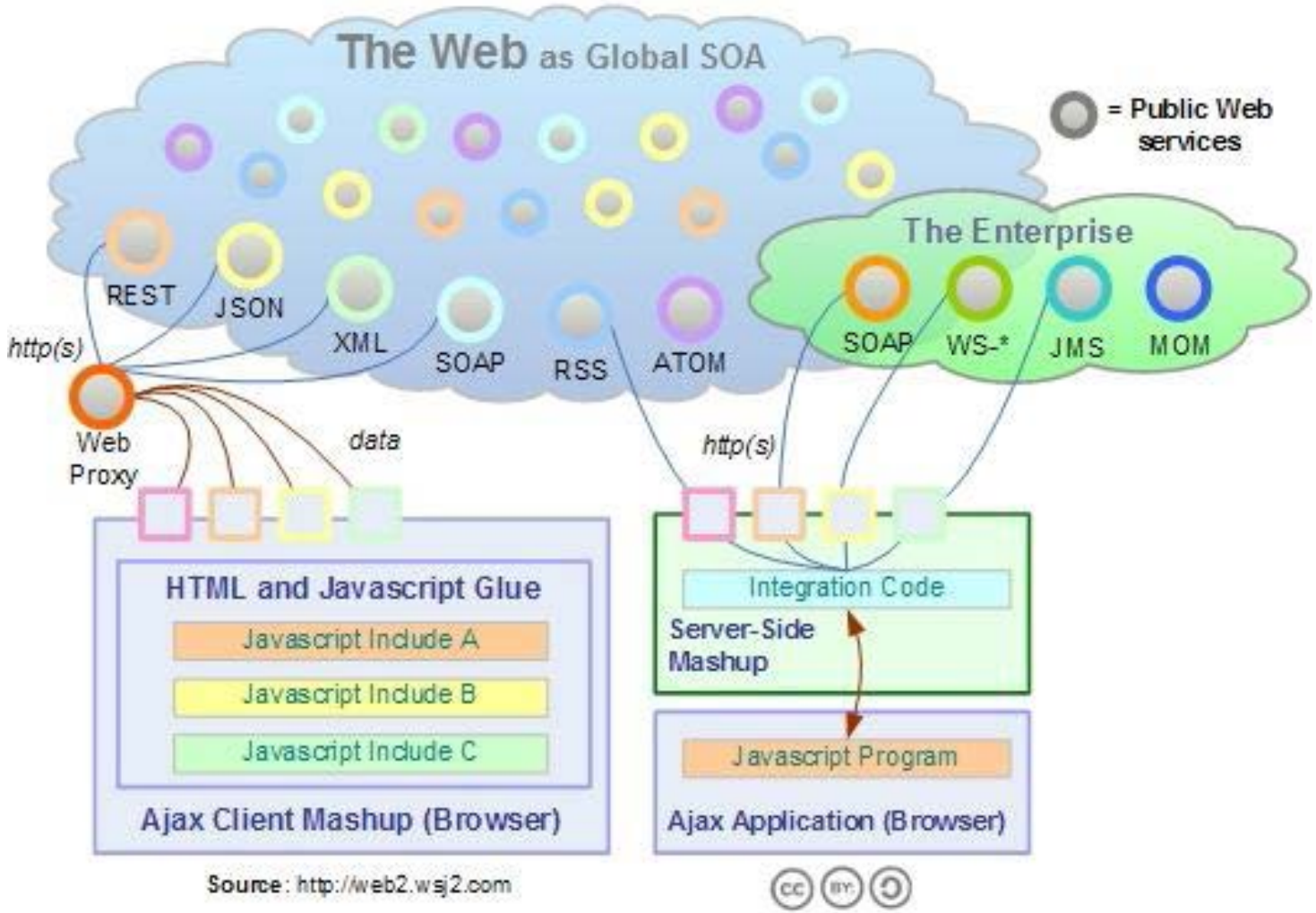


Mashup体系结构

- **Source**
  - **API/content providers**
  - **Web Protocols: REST, Web Services, RSS/ATOM**
  - **Screen Scraping**
- **The mashup site**
  - **This is where the mashup logic resides, it is not necessarily where it is executed**
  - **Server-side: Dynamic content aggregation**
  - **Client-side: Client side scripting**
- **The client's Web browser**
  - **This is where the application is rendered graphically and where user interaction takes place**

Web Mashup Styles

In-Browser | Server-side



Source: http://web2.wsj2.com

结 束!